Honeywell

5X80 Series

Software Development Kit (SDK) for 5080, 5180, and 5380 Decoded Miniature Image Scan Engines

User's Guide

Disclaimer

Honeywell International Inc. (“HII”) reserves the right to make changes in specifications and other information contained in this document without prior notice, and the reader should in all cases consult HII to determine whether any such changes have been made. The information in this publication does not represent a commitment on the part of HII.

HII shall not be liable for technical or editorial errors or omissions contained herein; nor for incidental or consequential damages resulting from the furnishing, performance, or use of this material.

Web Address: www.honeywellaidc.com

Table of Contents

Chapter 3 - Enumerated Types and Definitions

Chapter 4 - Structures and Mask Definitions

Chapter 5 - OEM-Configurable SDK Functionality

Chapter 6 - Program Samples

Chapter 7 - Customer Support

1 Introduction

The 5X10/5X80 Software Development Kit (5X00 Series) provides a set of libraries, tools, and sample source code to help software developers create an interface between their host system and a Honeywell miniature image scan engine. The 5X00 Series consists of:

- The API Definition and Documentation
- API Libraries
- Sample Code

Features of the 5X00 Series

- The 5X00 Series contains software libraries that interact with image/data capture engines using a documented API (Application Programming Interface). The API functions are defined on a higher level so they can be easily understood and integrated into your applications, so you don't have to learn minute details of the engine interface protocol. You simply compile your code with the library header files and link in the library for your platform. Afterward, all engine functionality is at your disposal.
- The image/data capture engine is easily integrated into a variety of host platforms.
- The 5X00 Series captures images and returns them as unformatted data, or as one of the standard file formats (BMP, TIFF, and JPG). Captured images can then be saved to disk and easily imported into a variety of common tools and applications.
- A single API is used for all Honeywell decoding engines. The libraries for all engines are identical for a given host platform. There are different libraries for each platform, but the API interface is the same for all of them, so you only need to learn a single API.
- Libraries are available for the Microsoft® Windows® family of operating systems. This includes both the Windows® CE operating system, Windows® 9x, and Windows NT® derivatives.
- Sample code is included that demonstrate how to use specific aspects of the 5X00 Series, as well as the buildable source and executable code for a demo application.
- The communication driver library is separate from the main engine API library.

Target Operating Systems for the 5X00 Series

The 5X00 Series is designed for use with the following operating systems:

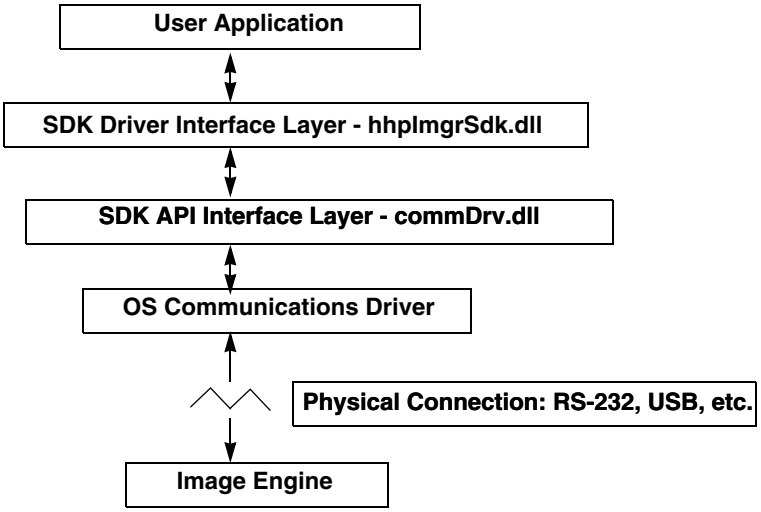
- Windows®CE versions WinCE 4.2, WinCE 5.0, Windows® Pocket PC 2000, and Pocket PC 2002 supporting the following processors:

  Pocket PC 2000ARM, MIPS, SH3
  PocketPC 2002ARM
  PocketPC 2003ARMV4
  CE.Net Standard SDK ARMV4, ARMV4I, ARMV4T, SH3, SH4, X86

- Windows® 9x, Windows NT® 4.0, Windows® 2000 and Windows® XP

Interface Diagram

The following diagram shows the interface between the 5X00 Series and the Image Engine:



5X00 Series Library Files

The SDK API and SDK communications layers are provided in the dynamic link libraries, hhpImgrSdk.dll and commDrv.dll, respectively. The library link file hhpImgrSdk.lib and the include files hhpImgrSdk.h, hhpSymCfg.h and hhpImgrCfg.h are also provided. In order to use the 5X00 Series, you must include hhpImgrSdk.h in any source files that call the SDK functions. The library include files must be in the include path for the application's project. This means that the files must either be in the source file build directory, or in the developer's Studio include path. Also, the library link file, hhpImgrSdk.lib, must be added to the application project link list and link path. Since the .dll and .lib files are operating system and processor type dependent, care must be taken to use the proper files for the chosen target environment.

SDK Library File	*Where Resides*	*Function*
commDrv.dll	Target Device Windows Folder	Sends commands from API to the appropriate OS driver. Handles low level communications protocols.
hhpImgrSdk.dll	Target Device Windows Folder	Contains the exported SDK API. Formats requests into imager commands, then calls commDrv layer.
hhpImgrSdk.lib	Project Link Path	Library exports file. Allows your application to link without actually including the SDK code at link time.
hhpSymCfg.h	Project Include Path	Definitions, structures, and enumerated types related to symbology setup. Automatically included when hhpImgrSdk.h is included.
hhpImgrCfg.h	Project Include Path	Definitions, structures, and enumerated types for imaging as well as all other imager configurations, except communications. Automatically included when hhpImgrSdk.h is included.
hhpImgrSdk.h	Project Include Path	Include file that must be included in application code. Contains definitions, structures, and enumerated types for communication configuration, as well as the SDK error codes and SDK function API prototypes.
OemDll.h	Project Include Path	Only necessary if you create a helper dll for the SDK to provide access to the imager hardware sleep and trigger lines.

5X00 Series API Library Summary

The following is a summary of the API functions. The full description of each function is found on the page noted.

Core Functions

Core Function	Summary	Page
Error Management Function		
hhpGetErrorMessage	Returns a descriptive text string for the specified SDK error code.	2-4
Connection Functions		
These functions open, close, and verify a connection to an imager.		
hhpConnect	Establishes and initializes a connection to the imager at the specified port and connection settings.	2-3
hhpDisconnect	Closes down the open imager connection.	2-3
hhpEngineConnected	Checks if the connection to the imager is valid.	2-4
hhpNamedConnect	Establishes and initializes a connection to the imager at the specified driver name and connection settings.	2-5
Asynchronous Notification and Control Functions		
Image and barcode capture can be either blocking (synchronous) or non-blocking (asynchronous). In blocking (synchronous) mode, the SDK function call does not return until the barcode or image is received, the request times out, or an error occurs. In non-blocking (asynchronous) mode, the capture call returns immediately. Your application is notified of the completion when either a barcode or an image was received, the time-out for the call was reached, or an error was detected. You can specify which notification methods you wish to receive. Your application can subscribe to one or more of the following notification methods: Windows Event, Windows Message, and/or Callback Function. When notification is received, your application can call hhpGetAsyncResult (see page 2-4) to retrieve the return code as well as the image, barcode, or text data. The asynchronous interface is also the manor in which imager-initiated barcode capture data, such as from a hardware trigger, is returned. There is also a cancel function (see hhpCancelIo on page 2-2) that allows you to cancel any ongoing operation. You should be aware that when the callback function method is used, any processing done during the callback is run within the context of the SDK's asynchronous read thread. This means that the SDK is unable to receive images or barcodes until the callback returns.		
hhpCancelIo	Cancels any synchronous or asynchronous I/O in process.	1-3
hhpGetAsyncResult	Retrieves the results (image/barcode, etc.) of an asynchronous I/O event.	2-4
hhpSetAsyncMethods	Allows the application to select how it wishes to be notified on completion of an asynchronous I/O event.	2-10
Imaging and General Configuration Functions		
The imaging and configuration functions provide a simplified API for modifying the imager setup, image/barcode capture configuration, and symbology configuration. In order to limit the number of functions a developer must master, the design philosophy is to allow configuration control using only a small number of setup functions. The imager configuration is broken down into functional groups with structures containing the configurable items for each. Individual configuration items are specified within structures by use of a bit field mask. In this way, single configuration items can be read or written using minimal communication traffic. There are functions for reading and writing parts or all of the HHP_CONFIG imager configuration structure as well as writing the setup/configuration for individual symbologies. If the specified symbology is not available in the imager's version of the symbol decoder, (e.g., Data Matrix in a linear and PDF417 decoder), the symbology functions return RESULT_ERR_UNSUPPORTED. Finally, to facilitate easy configuration management from device to device and application to application, the 5X00 Series also provides methods for retrieving and setting the whole imager configuration as a single stream so it can be saved to disk and restored at a later time.		
hhpReadConfigItem	Retrieves a single configuration group or whole imager configuration from the imager.	2-6
hhpReadConfigStream	Retrieves the current whole imager configuration as a single buffer. This buffer can be saved to a file and later restored.	2-7

Core Functions (Continued)

Core Function	Summary	Page
hhpReadImagerCapabilities	Retrieves the imager settings (fixed) for image size, image bit depth, and maximum message length.	2-8
hhpReadEngineInfo	Reads information about the image engine contained in the image engine PSOC.	2-8
hhpSetConfigItemToDefaults	Sets a selected symbology, or all symbology configurations, to their default values.	2-10
hhpWriteConfigItem	Writes some or all of the configuration items for a single configuration group or for all configuration groups, with the exception of version and communication groups.	2-12
hhpWriteConfigStream	Writes an entire data stream of programmable parameters from a previous call to hhpReadConfigStream.	2-13
Symbology Configuration Functions		
These functions allow you to read and set the symbology configurations.		
hhpEnableDisableSymbology	Enables or disables a single symbology, or all symbologies.	2-3
hhpReadSymbologyConfig	Retrieves the current or default symbology configuration for a specified symbology, or for all symbologies.	2-8
hhpReadSymbologyRangeMaxMin	Returns the specified symbology range maximum and minimum values.	2-9
hhpSetSymbologyDefaults	Defaults a single symbology, or all symbologies.	2-11
hhpWriteSymbologyConfig	Writes some or all of a single symbology, or all symbologies' configuration items.	2-14
Barcode Capture Functions		
The 5X00 Series captures barcodes from imagers that have hardware triggers or some other non-SDK initiated barcode captures without having to poll the imager to see if there is any data to read. This allows the imager to be put into low power mode without having to wake up to answer the polling message. All barcode result strings are returned in TCHAR arrays, which, if running on a WinCE device or if using a Unicode Desktop build, are 2 bytes per character. You can specify a Unicode code page other than the default ANSI code page (CP_ACP).		
hhpCaptureBarcode	Initiates a synchronous (wait for finish before returning from call) or asynchronous (return immediately) barcode capture. Decoded data returned is translated by code page and locale.	2-2
hhpSetBarcodeDataCodePage	Specifies the code page used to convert the barcode characters to Windows text. The default is the ANSI code page.	2-10
hhpCaptureRawBarcode	Initiates a synchronous (wait for finish before returning from call) or asynchronous (return immediately) barcode capture. Decoded data returned is unmodified 8 bit (ASCII) data.	2-2
Image Capture Functions		
The image capture functions provide both synchronous and asynchronous operation. A synchronous capture is specified by setting the bWait parameter of hhpAcquireImage or hhpGetLastImage to TRUE. For synchronous operation, the function will not return until an image has been captured and transferred (hhpAcquireImage), just transferred (hhpGetLastImage), or an error has occurred. Asynchronous captures are specified by setting bWait to FALSE. The function call returns immediately and the caller is notified on request completion as long as at least one of the event notification methods has been enabled. You can receive transfer progress updates by Windows messages or by providing a pointer to a DWORD. Both hhpAcquireImage and hhpGetLastImage allow the caller to override the current imager transfer configuration in the imager.		
hhpAcquireImage	Initiates a synchronous (wait for finish before returning from call) or asynchronous (return immediately) image capture. The image acquisition and transfer parameters can also be specified.	2-1

Core Functions (Continued)

Core Function	Summary	Page
hhpGetLastImage	Initiates transfer of the last image captured. (This includes images captured during barcode scan.) The call can be made synchronously or asynchronously, and the transfer parameters can be specified.	2-4
Intelligent Imaging (Signature Capture) Functions		
Intelligent imaging is barcode capture combined with an image window capture. The image window is cut from the same image used to capture a barcode. This is how the SDK provides the ability to capture a signature associated with a barcode. In fact, a successful barcode capture is required before the intelligent image window is sent by the imager. You can specify whether the image is returned grayscale or black and white.		
hhpAcquireIntelligentImage	Barcode capture combined with an image window capture.	2-1
hhpRawAcquireIntelligentImage	Captures portion of the image in which a barcode is decoded.	2-5
Miscellaneous Functions		
hhpSendActionCommand	Turns on illumination LEDs, Aimers, (and sound beeper for imagers that have one) outside the image or barcode capture.	2-9
hhpSendMessage	Sends menu commands to the imager.	2-9
hhpUpgradeFirmware	Upgrades the current imager firmware with a new firmware file.	2-11
hhpSetHardwareLineDllFileName	Allows you to specify the name of an OEM dll file. This file can contain some or all of the OemDll exports that provide support for hardware trigger and low power mode hardware lines.	2-11

Data Types, Structures, and Enumerated Types

The 5X00 Series API uses structures (see Structures and Mask Definitions beginning on page 4-1) and enumerated types (see Enumerated Types and Definitions beginning on page 3-1) extensively . The definitions are in the include files in the 5X00 Series package. All 5X00 Series-specific structures have a dwStructSize member that must be set to sizeof(struct name). This insures that the structure being passed to a given function is the structure type expected by the function and, if writing is done to the structure, that the structure size boundary is not exceeded. Furthermore, all imager configuration structures (except the all inclusive structure HHP_CONFIG) have a DWORD member dwMask. The mask allows you to specify only certain members within a structure. Set the mask value by ORing together the appropriate masks for the given structure for the particular items within the structure that should be read/written. Samples of programs that demonstrate this can be found in Program Samples beginning on page 6-1. This technique is also used by Microsoft® in their Windows® SDK (for example, see Windows® SDK structure CHARFORMAT).

The following Windows® data types are included for clarity.

Note: A "P" in front of a data type means a pointer to the type.

Windows Data Types	
BOOL	32 bit signed integer used by most Microsoft SDK functions in place of a true Boolean.
BYTE	8 bit unsigned variable.
DWORD	32 bit unsigned integer variable.
HANDLE	A Windows WIN32 handle type. Returned from opening files, creating events, semaphores, or mutexes.
HWND	A Windows handle to an application window.
PVOID	32 bit unsigned integer that points to void data type (generic pointer).
TCHAR	OS-dependent character variable. 16 bit for Unicode systems, otherwise 8 bits.
WORD	16 bit unsigned integer variable.

Windows Data Types (Continued)	
HHP_EVENT_CALLBACK	Pointer to a callback function (see hhpImgrSdk.h) called in response to the completion of an asynchronous 5X00 Series function call or event. (See example on page 6-4.)
SDK Enumerated Types	
Beep Options	Enumeration (not an enumerated type) that can be used with an hhpSendActionCommand.
Compression_t	Enumerated type for specifying the form of compression (if any) to use when transferring an image from the imager to the SDK. See Compression Mode Formats on page 3-4.
ConfigItems_t	Enumerated type to specify that the configuration structure is being sent to the Read/Write config item functions.
DECODE_METHOD	Enumerated type of decode methods available to decode symbols.
DECODER_TYPE	Enumeration of types of decoders and, by extension, what symbologies can be decoded.
EngineType_t	Describes the connection, the imager, and the type of engine.
FileFormat_t	Enumerated type for specifying the format of the data returned in the HHP_IMAGE structure.
HP_ACTION	Enumerated type whose items describe which imager command functionality (beeper, aimers, lights) is to be acted upon.
HHP_AIMER_MODES	Enumerated type to specify the aimer mode.
HHP_AUTOEXPOSURE	Enumerated type to specify whether the imager tries to auto adjust the image exposure and, if so, how.
HHP_BAUD_RATE	Enumerated type of the supported baud rates for serial devices. *Note: A special driver is required if a baud rate greater than 115200 is selected.*
HHP_BEEPER_VOLUME	Enumerated type to select the imager beeper volume when sounding the beeper. This structure is ignored for products that do not have a beeper.
HHP_CONNECT_TYPE	Enumerated type used in hhpConnect to specify the connection type and connection port where the imager is connected.
HHP_DATA_BITS	Enumerated type for number of serial data bits.
HHP_DUTY_CYCLE	Enumerated type to specify the behavior of the illumination and aimer LEDs during image capture.
HHP_EVENT_TYPE	Enumerated type that describes the type of asynchronous event being reported.
HHP_FRAME_RATE	Enumerated type to select the image capture frame rate. Only valid when no auto exposure is selected.
HHP_GAIN	Enumerated type to select the image capture gain. This type is only valid when no auto exposure is selected.
HHP_PARITY	Enumerated type for serial parity.
HHP_SEQ_MODES	Enumerated type to specify the sequence acquisition mode.
HHP_STOP_BITS	Enumerated type for number of serial stop bits.
HHP_SYS_SPEED	Enumerated type to specify the rate (in Mhz) at which all components other than the CPU are to be clocked.
HHP_TRIG_MODES	Enumerated type to specify the trigger mode.
OCRDirection_t	Enumerated type for setting the text direction for OCR decoding.
OCRMode_t	Enumerated type for setting the font for OCR symbology decoding.
On Off	Enumeration (not an enumerated type) that can be used with hhpSendActionCommand.
Result_t	Enumerates the function result codes returned by the SDK functions. See Error Codes on page 3-1.
SetupType_t	Enumerated type for specifying whether a read configuration item call should return the current settings or the imager default setting.
Symbology ID enumeration	Enumerates all the available symbologies supported in the imager decoder. Non-data type used in the symbology configuration functions.

SDK Structure Types

A full description of the SDK structure types can be found in Enumerated Types and Definitions beginning on page 3-1.

Note: Important: Make sure to set the structure members dwStructSize and dwMask.

Individual Symbology Structures	
All symbologies, except OCR, use either SymFlagsOnly_t or SymFlagsRange_t for configuration. There is a define for each symbology (except OCR) that points to one of these two structures.	
SymFlagsOnly_t	Structure for symbologies that don't have minimum and maximum data lengths.
SymFlagsRange_t	Structure for symbologies that have minimum and maximum data lengths.
SymCodeOCR_t or OCR_T	Structure to configure OCR symbology.
Configuration Structures	
HHP_BEEPER	Configures whether the beeper sounds on power up, decode, or command processing. This is ignored if the imager does not have a beeper.
HHP_CONFIG	Super structure whose members are all the other configuration structures.
HHP_DECODE_MSG	Returns decoded, code page and locale-translated data output from the SDK.
HHP_DECODER_CONFIG	Configures decoding behavior other than symbology setup.
HHP_ENGINE_INFO	Structure used by the hhpReadEngineInfo (page 2-8) call that returns information about the image engine. (5X80 engines with PSOC only.)
HHP_IMAGE	Returns image data from the SDK. It also specifies the format in which the image data is returned.
HHP_IMAGE_ACQUISITION	Specifies how images are captured by the imager. This includes gain, exposure, frame rates, and illumination.
HHP_IMAGE_TRANSFER	Specifies how images are shipped from the imager to the SDK. This includes image processing items such as cropping, subsampling, histogram stretching, and transfer compression.
HHP_IMAGER_CAPS	Retrieves the fixed characteristics of the imager: full image size, capture bits per pixel, and maximum text/barcode message size (sent from imager). Structure in which hhpReadImagerCapabilities returns requested capabilities information.
HHP_INTEL_IMG	Specifies the location and size of the image returned as part of an intelligent image capture. The location information is specified in minimum bar widths of the barcode portion of the intelligent image.
HHP_POWER_SETTINGS	Configures the power management options of the imager.
HHP_RAW_DECODE_MSG	Returns raw (8 bit ASCII) decoded data output from the SDK.
HHP_SERIAL_PORT_CONFIG	Specifies serial port configuration for serial imagers.
HHP_SEQUENCE	Specifies how the sequence acquisition mode is configured.
HHP_SYM_CONFIG	Contains a list of all the structures for all the symbologies.
HHP_TEXT_MSG	Returns non-decoded data output from the SDK. The data is translated by code page and locale.
HHP_TRIGGER	Configures the trigger mode of the imager.
HHP_VERSION_INFO	Queries imager and SDK software revisions.

2 API Function Descriptions

The following is an alphabetic listing of each API function with its complete description and a prototype for each function. All API functions (with the exception of hhpEngineConnected (page 2-4) return a result code of type Result_t. See Error Codes on page 3-1 for the result code values.

hhpAcquireImage

This function causes the imager to capture an image and transfer it to the host. Values to be used from the structures are specified by setting the appropriate bit mask for each item in the structure's mask member.

hhpAcquireImage(
PHHP_IMAGE pImg,
PHHP_IMAGE_TRANSFER pImgTrans,
PHHP_IMAGE_ACQUISITION pImgAcqu,
BOOL bWait
)

Parameter	Description
pImg	Pointer to an HHP_IMAGE structure if bWait is TRUE. If bWait is FALSE, the parameter is ignored and should be NULL.
pImgTrans	Optional pointer to an HHP_IMAGE_TRANSFER structure. This structure overrides (just for this call) the current imager configuration, and specifies the pixel subsample, cropping rectangle, transfer compression type, compression factor (for JPEG lossy transfer), and progress notification method. If this parameter is NULL, the current imager configuration settings are used except for the progress notification methods that must be specified for each call if notification is desired.
pImgAcqu	Optional pointer to an HHP_IMAGE_ACQUISITION structure. This structure overrides (just for this call) the current imager configuration, to specify and configure the image capture method (type of autoexposure control or manual mode). If this parameter is NULL, the current imager configuration settings are used.
bWait	If TRUE, do not return until the image is received or an error occurs. If FALSE, return immediately. One of the event notification methods must be enabled to receive notification on completion. (See hhpSetAsyncMethods on page 2-10.)

hhpAcquireIntelligentImage

The location of the window of interest must be provided in units of minimum barcode widths. This allows the imager to grab the same physical window, no matter how far the imager is from the page. The resultant image window is always squared with the X and Y axis of the returned image, so even if the barcode page is rotated relative to the imager, the resultant image appears square to the image edges.

There is only one intelligent image call that supports both synchronous and asynchronous capture. If synchronous capture is used, all members of this structure must be valid. If asynchronous capture is used, you will receive HHP_INTELIMG_BARCODE_EVENT for the barcode data, and HHP_INTELIMG_IMAGE_EVENT for the image data. The barcode data is returned in a normal barcode structure (HHP_DECODE_MSG), while the intelligent image data is returned in an HPP_IMAGE structure.

Note: Since the HHP_INTEL_IMG structure requires that image offsets and size be specified in barcode units, the HHP_INTEL_IMG structure has a size member that allows you to specify (in pixels) the maximum allowable width and height for the returned image.

hhpAcquireIntelligentImage(
PHHP_INTEL_IMG pIntelImg,
PHHP_DECODE_MSG pDecodeMsg,
DWORD dwTimeout,
PHHP_IMAGE pImg,
BOOL bWait
)

Parameter	Description
pIntelImg	Pointer to an HHP_INTEL_IMG structure that contains the setup parameters describing the location of the intelligent image relative to the barcode.
pDecodeMsg	Pointer to an HHP_DECODE_MSG structure if bWait is TRUE. If bWait is FALSE, the parameter is ignored and should be NULL. The intelligent image barcode information is returned here.
dwTimeout	Maximum time (in milliseconds) to attempt to decode before declaring a no decode.
pImg	Pointer to an HHP_IMAGE structure if bWait is TRUE. If bWait is FALSE, the parameter is ignored and should be NULL.
bWait	If TRUE, do not return until the image is received or an error occurs. If FALSE, return immediately. One of the event notification methods must be enabled to receive notification on completion. (See hhpSetAsyncMethods on page 2-10.)

hhpCancelIo

Cancels the current barcode or image capture.

hhpCancelIo(
void
)

hhpCaptureBarcode

This function causes the imager to capture images and attempt to decode them. Decoded data returned is translated by code page and locale. Barcode capture can be synchronous or asynchronous. Synchronous capture is specified by setting the bWait parameter hhpCaptureBarcode to TRUE. In this case, the function will not return until a barcode is read, an error occurs, or the decode timeout is reached. Asynchronous capture is specified by setting the bWait parameter hhpCaptureBarcode to FALSE, or whenever a barcode capture is initiated other than by the 5X00 Series (e.g., from a hardware trigger). In order to be notified of an asynchronous transfer, you must enable at least one of the notification methods (see hhpSetAsyncMethods on page 2-10).

hhpCaptureBarcode(
PHHP_DECODE_MSG pDecodeMsg,
DWORD dwTimeout,
BOOL bWait
)

Parameter	Description
pDecodeMsg	Pointer to an HHP_DECODE_MSG structure if bWait is TRUE. If bWait is FALSE, the parameter is ignored and should be NULL. (HHP_DECODE_MSG will be passed to hhpGetAsyncMethods() call instead.) The function returns immediately, and you are notified when symbols are decoded, an error occurs, or decoding times out (no barcode) using the specified event notification method(s).
dwTimeout	Maximum time (in milliseconds) to attempt to decode before declaring a no decode. A value of CURRENT_DECODE_TIMEOUT specifies that the timeout is whatever is currently set on the imager. A value of 0 indicates no timeout.
bWait	If TRUE, wait for capture to complete before returning. If FALSE, one of the event notification methods must be enabled to receive notification upon completion.

hhpCaptureRawBarcode

This function causes the imager to capture images and attempt to decode them. Decoded data returned is unmodified 8 bit ASCII data. Barcode capture can be synchronous or asynchronous. Synchronous capture is specified by setting the bWait parameter hhpCaptureRawBarcode to TRUE. In this case, the function will not return until a barcode is read, an error occurs, or the decode timeout is reached. Asynchronous capture is specified by setting the bWait parameter hhpCaptureRawBarcode to FALSE, or whenever a barcode capture is initiated other than by the 5X00 Series (e.g., from a hardware trigger). In order to be notified of an asynchronous transfer, you must enable at least one of the notification methods (see hhpSetAsyncMethods on page 2-10).

hhpCaptureBarcode(
PHHP_RAW_DECODE_MSG pDecodeMsg,
DWORD dwTimeout,
BOOL bWait
)

Parameter	Description
pDecodeMsg	Pointer to an HHP_RAW_DECODE_MSG structure if bWait is TRUE. If bWait is FALSE, the parameter is ignored and should be NULL. (HHP_DECODE_MSG will be passed to hhpGetAsyncMethods() call instead.) The function returns immediately, and you are notified when symbols are decoded, an error occurs, or decoding times out (no barcode) using the specified event notification method(s).
dwTimeout	Maximum time (in milliseconds) to attempt to decode before declaring a no decode. A value of CURRENT_DECODE_TIMEOUT specifies that the timeout is whatever is currently set on the imager. A value of 0 indicates no timeout.
bWait	If TRUE, wait for capture to complete before returning. If FALSE, one of the event notification methods must be enabled to receive notification upon completion.

hhpConnect

This function opens a connection to an imager. The connection must be closed by calling hhpDisconnect(). The caller can verify that the imager is connected by calling hhpEngineConnected().

Opens the selected communications port and establishes connection with the imager and starts the read data thread.

hhpConnect(
HHP_CONNECT_TYPE connectType,
PVOID pStruct
)

Parameter	Description
connectType	Describes the type of connection and hardware port, e.g., HHP_COM1, HHP_COM2, HHP_COM3.
pStruct	An optional structure that contains setup information for the hardware port, or NULL. PHHP_SERIAL_PORT_CONFIG is used to configure a serial port connection.

hhpDisconnect

Closes the communications port and stops the read data thread.

hhpDisconnect(
void
)

hhpEnableDisableSymbology

Enables/disables an individual symbology or all symbologies.

hhpEnableDisableSymbology(
int nSymId,
BOOL bEnable
)

Parameter	Description
nSymId	One of the symbology enumerated types, e.g., SYM_CODE39, SYM_OCR, or SYM_ALL to enable/disable all symbologies.
bEnable	TRUE to enable symbology, FALSE to disable symbology.

hhpEngineConnected

This function determines whether the imager is connected. This function checks to to see if the imager has lost power (due to the host going into a suspended state), or if the imager has been removed.

hhpEngineConnected(
void
)

hhpGetAsyncResult

Retrieves the data from the last signal event (image/barcode capture). This function can be called with pResultStruct set to NULL to obtain the event type. This is useful when the notification method is a Windows event.

Result_t **hhpGetAsyncResult**(
hhpEventType_t *pEventType,
PVOID pResultStruct
)

Parameter	Description
hEventType	Type of data causing the event notification. The valid values are: HHP_BARCODE_EVENT HHP_IMAGE_EVENT HHP_TEXT_ MSG_EVENT HHP_INTELIMG_BARCODE_EVENT HHP_INTELIMG_IMAGE_EVENT
pResultStruct	An HHP_DECODE_MSG, HHP_IMAGE, or HHP_TEXT_MSG structure pointer, depending on the value of hEventType. This parameter can be NULL if just the event type is desired. This is of use when the Event Handle notification is used.

hhpGetErrorMessage

This function returns a text message describing the meaning of a Result_t error code. See Error Codes on page 3-1 for complete descriptions.

hhpGetErrorMessage(
Result_t nErrorCode,
PTCHAR ptcErrorMsg,
int nMaxChars
)

Parameter	Description
nErrorCode	Error code returned from one of the other 5X00 Series functions.
ptcErrMsg	TCHAR buffer to hold error message string.
nMaxChars	Maximum number of characters that can fit in ptcErrorMsg including NULL.

hhpGetLastImage

This function causes the imager to transfer the last image captured to the host. If bWait is TRUE, the function will not return until the image is fully received or an error occurs. If bWait is FALSE, the function returns immediately and you are notified when image transfer has completed or an error has occurred. pImgTrans is an optional parameter and can be NULL. Setting the appropriate bit mask for each item specifies active members of this structure. This function can be used to obtain the image from the last barcode capture attempt as well as the last image from an image capture attempt.

hhpGetLastImage(
PHHP_IMAGE pImg,
PHHP_IMAGE_TRANSFER pImgTrans,
BOOL bWait
)

Parameter	Description
pImg	Pointer to an HHP_IMAGE structure if bWait is TRUE. If bWait is FALSE, the parameter is ignored and should be NULL.
pImgTrans	Optional pointer to an HHP_IMAGE_TRANSFER structure. This structure overrides (just for this call) the current imager configuration, and specifies the pixel subsample, cropping rectangle, transfer compression type, compression factor (for JPEG lossy transfer), and progress notification method. If this parameter is NULL, the current imager configuration settings are used except for the progress notification methods that must be specified for each call if notification is desired.
bWait	If TRUE, do not return until the image is received or an error occurs. If FALSE, return immediately. One of the event notification methods must be enabled to receive notification on completion.

hhpNamedConnect

This function opens a connection to an imager. The connection must be closed by calling hhpDisconnect (page 2-3). The caller can verify that the imager is connected by calling hhpEngineConnected (page 2-4).

hhpNamedConnect(
PTCHAR ptcConnectName,
PVOID pStruct
)

Parameter	Description
ptcConnectName	The name of driver for the hardware port, e.g., "COM1:" or "\\.\COM12."
pStruct	An optional structure that contains setup information for the hardware port, or NULL. HHP_SERIAL_PORT_CONFIG is used to configure a serial port connection.

hhpRawAcquireIntelligentImage

Captures a portion of the image in which a barcode is decoded. The position of the image is specified relative to the center of the barcode. This function differs from hhpAcquireIntelligentImage (page 2-1) in that the barcode data is returned as a raw (untranslated) byte data array.

Note: Since the HHP_INTEL_IMG structure requires that image offsets and size be specified in barcode units, the HHP_INTEL_IMG structure has a size member that allows you to specify (in pixels) the maximum allowable width and height for the returned image.

hhpRawAcquireIntelligentImage(
PHHP_INTEL_IMG pIntelImg,
PHHP_RAW_DECODE_MSG pRawDecodeMsg,
DWORD dwTimeout,
PHHP_IMAGE pImg,
BOOL bWait
)

Parameter	Description
pIntelImg	Pointer to an HHP_INTEL_IMG structure that contains the setup parameters describing the location of the intelligent image relative to the barcode.
pRawDecodeMsg	Pointer to an HHP_RAW_DECODE_MSG structure if bWait is TRUE. If bWait is FALSE, the parameter is ignored and should be NULL. The intelligent barcode raw data (untranslated) is returned here.
dwTimeout	Maximum time (in milliseconds) to attempt to decode before declaring a no decode.
pImg	Pointer to an HHP_IMAGE structure if bWait is TRUE. If FALSE, the parameter is ignored and should be NULL.
bWait	If TRUE, do not return until the image is received, the decode timeout is reached, or an error occurs. If FALSE, return immediately. One of the event notification methods must be enabled to receive notification of completion (see hhpSetAsyncMethods on page 2-10).

hhpReadConfigItem

Reads the configuration items for one or all of the configuration structures found in the main 5X00 Series configuration structure HHP_CONFIG.

hhpReadConfigItem(
SetupType_t cfgType,
ConfigItems_t item,
PVOID pStruct
)

Parameter	Description	
cfgType	Use SETUP_TYPE_CURRENT for the current settings, or SETUP_TYPE_DEFAULT for the customer default settings.	
item	One of the members of the enumerated type ConfigItems_t. The valid values are:	
	BEEPER_CONFIG	Returns beeper control settings for devices with audible beepers. The settings include volume, beep on decode, and beep on reset.
	TRIGGER_CONFIG	Returns values for all trigger timeouts and current trigger mode.
	DECODER_CONFIG	Returns the decoder setup. This includes single decode, multiple decodes, print weight, centered decode, and aimer.
	POWER_CONFIG	Contains low power configuration settings: trigger mode, low power timeout, low power imaging, LED brightness, aimer LED settings, and system clock speed.
	VERSION_INFO	Returns the SDK, imager firmware, and imager boot code versions.
	SYMBOLOGY_CONFIG	Returns the enable and setup parameters for all symbologies. Individual symbologies can be read by calling hhpReadSymbologyConfig.
	SERIAL_PORT_CONFIG	If using a serial connection imager, this returns the serial port setup parameters of the imager.
	IMAGE_ACQUISITION	Returns the current imager settings for capture mode, manual exposure, manual gain, manual frame rate, target white value, target window, aimer and illumination duty cycle, and triggering mode (hardware trigger dependent).
	IMAGE_TRANSFER	Returns the current image transfer protocols, such as cropping window, pixel subsample, and transfer compression.
	SEQUENCE_CONFIG	Returns the sequence mode enable/disable state and the data sequence command string.
	ALL_CONFIG	All of the above except serial port config.
pStruct	Pointer to the appropriate structure based on parameter “item:”	
	HHP_BEEPER	
	HHP_TRIGGER	
	HHP_DECODER_CONFIG	
	HHP_POWER_SETTINGS	
	HHP_VERSION_INFO	
	HHP_SYM_CONFIG	
	HHP_SERIAL_PORT_CONFIG	
	HP_IMAGE_ACQUISITION	
	HHP_IMAGE_TRANSFER	
	HHP_SEQUENCE	
	HHP_CONFIG	

hhpReadConfigStream

Reads the full imager configuration as a single stream of data into a buffer. The buffer contains all the configuration items in an ASCII stream so that it can be written to a disk for storage. No interpretation is done on the data stream, therefore, the data stream contains both read only, and read/write data.

hhpReadConfigStream(
PBYTE puchCfgStream,
int nMaxLen,
PINT pnBytesReturned
)

Parameter	Description
puchCfgStream	Buffer to hold the raw imager configuration stream.
nMaxLen	Maximum number of bytes that fit in buffer puchCfgStream.
pnBytesReturned	Pointer to an integer where the number of bytes returned in puchCfgStream is placed.

hhpReadEngineInfo

Reads information about the image engine contained in the image engine PSOC. This call is only valid for imagers that have a PSOC. If the attached imager is not an 5X80 image engine, or the engine does not have a PSOC, the function returns the error code RESULT_ERR_UNSUPPORTED.

hhpReadEngineInfo(
PHHP_ENGINE_INFO pEngInfo
)

Parameter	Description
pEngInfo	Pointer to an engine information structure in which the engine information is returned. The dwStructSize member must be set to size (HHP_ENGINE_INFO) before making the call.

hhpReadImagerCapabilities

Returns the fixed imager capabilities, such as imager bits per pixel or image capture width and height.

Note: As with all other HHP structures, the dwStructSize member of the structure must be set before calling this function. (Set to sizeof (HHP_IMAGER_CAPS).)

hhpReadImagerCapabilities(
HHP_IMAGER_CAPS pImgrCaps
)

Parameter	Description
pImgrCaps	Pointer to the HHP_IMAGER_CAPS structure.

hhpReadSymbologyConfig

Reads configuration items for a single symbology or for all symbologies. Individual items to be read are specified by adding the appropriate mask bit (OR it) to the mask member of the structure to which it belongs. Only items whose bits are set are read; all other items are ignored.

hhpReadSymbologyConfig(
SetupType_t cfgType,
int nSymbol,
PVOID pvSymStruct
)

Parameter	Description
cfgType	Use SETUP_TYPE_CURRENT for the current settings, or SETUP_TYPE_DEFAULT for the customer default settings.
nSymbol	One of the symbology enumerated types, e.g., SYM_CODE39, SYM_OCR, or SYM_ALL to read all symbologies.
pStruct	Pointer to the appropriate structure based on nSymbol, e.g., CODE39_T, OCR_T, or HHP_SYM_CONFIG if all symbologies.

hhpReadSymbologyRangeMaxMin

Returns the specified symbology range maximum and minimum values. If a symbology has no range values, the function returns -1 for the minimum and maximum values.

hhpReadSymbologyRangeMaxMin(
int symbol,
PLONG pnRangeMin,
PLONG pnRangeMax
)

Parameter	Description
int	The enumerated symbology types, eg., SYM_CODE39, SYM_PDF417, or SYM_ALL to read the max/min range for all symbologies.
pnRangeMin	A LONG pointer to hold the minimum range value for single symbologies, or a LONG array of size NUM_SYMBOLOGIES if SYM_ALL specified. The min value will be -1 if the symbology does not support a minimum length value.
pnRangeMax	A LONG pointer to hold the maximum range value for single symbologies, or a LONG array of size NUM_SYMBOLOGIES if SYM_ALL specified. The max value will be -1 if the symbology does not support a maximum length value.

hhpSendActionCommand

This command allows the application to modify some of the imager hardware states. The items that can be modified include turning the illumination LEDs on/off, turning the aimer LEDs on/off, or causing the device's beeper to beep/double beep.

hhpSendActionCommand(
HHP_ACTION actionCmd,
int nVal)

Parameter	Description
actionCmd	One of the values of enum HHP_ACTION (HHP_AIMER_CMD, HHP_ILLUMINATION_CMD, or HHP_BEEP_CMD).
nVal	HHP_ON/HHP_OFF for illumination or aimers HHP_SINGLE_BEEP/HHP_DOUBLE_BEEP for beeper.

hhpSendMessage

The SDK API provides access to almost all of the imager command set. hhpSendMessage allows applications to send menu (imager) commands directly to the imager (both wrapped and unwrapped) and to receive the actual uninterpreted imager response. This command allows a developer to send debug commands to the imager.

hhpSendMessage(
PBYTE puchMsg,
int nLen,
BOOL bSendRaw,
PBYTE puchReply,
int nLenToRead,
PINT pnRetLen
)

Parameter	Description
puchMsg	Command sent to the imager with or without command wrapper. If no wrapper, set bSendRaw to TRUE.
nLen	Number of bytes to send (in puchMsg).
bSendRaw	If TRUE, the SYN M CR command wrapper is NOT added to the command before sending it to the imager. If FALSE, the command is sent withSYN M CR command wrapper.
puchReply	Buffer to hold imager response. Can be NULL if no response required.
nLenToRead	Number of bytes to read from imager in response. 0 if no response.
pnRetLen	Pointer to number of bytes returned in pnRetLen. NULL if no response required.

hhpSetAsyncMethods

hhpSetAsyncMethods sets the methods by which the caller wishes to be notified upon receipt of a barcode or image.

hhpSetAsyncMethods(
HANDLE hEventHandle,
HWND hWndHandle,
HHP_EVENT_CALLBACK EventCallback
)

Parameter	Description
hEventHandle	Handle to a Windows Event. The event should specify manual reset.
hWndHandle	Handle to the application window that should receive the SDK defined message WM_HHP_EVENT_HWND_MSG. The message parameters are: WPARAM The event type (hhpEventType_t) LPARAM The number of bytes received
EventCallback	Callback function of type HHPEVENTCALLBACK, which is BOOL CALLBACK name (EventType_t,DWORD).

hhpSetBarcodeDataCodePage

This function changes the code page used when translating the decoded data from a string of bytes to Unicode. The default value is CP_ACP (ANSI code page). There is no error checking on the values sent to this function, so you must determine whether or not a code page is valid on the given system.

hhpSetBarcodeDataCodePage(
DWORD dwCodePage
)

Parameter	Description
dwCodePage	Code page to use when converting from BYTE string to Unicode. The only 2 code pages that are valid are CP_ACP and CP_OEMCP.

hhpSetConfigItemToDefaults

Defaults a configuration group or individual group structure items.

hhpSetConfigItemToDefaults(
ConfigItems_t item
)

Parameter	Description	
item	One of the members of the enumerated type ConfigItems_t. The valid values are:	
	BEEPER_CONFIG	Resets HHP_BEEPER structure settings to their defaults.
	TRIGGER_CONFIG	Resets HHP_TRIGGER structure settings to their defaults.
	DECODER_CONFIG	Resets HHP_DECODER structure settings to their defaults.
	POWER_CONFIG	Resets HHP_POWER_SETTINGS structure settings to their defaults.
	SYMBOLOGY_CONFIG	Resets HHP_SYM_CONFIG structure settings to their defaults.
	SERIAL_PORT_CONFIG	Resets HHP_SERIAL_PORT_CONFIG structure settings to their defaults. Connection to the imager is maintained.
	IMAGE_ACQUISITION	Resets HHP_IMAGE_ACQUISITION structure settings to their defaults.
	IMAGE_TRANSFER	Resets HHP_IMAGE_TRANSFER structure settings to their defaults (not progress notification methods).
	ALL_CONFIG	Resets all of the above.

hhpSetHardwareLineDllFileName

The SDK API provides the ability to provide OEM device-dependent extensions to support the imager hardware sleep lines, and hardware trigger and/or special COM port driver configuration/initialization. The definitions and function prototype are located in the header file OemDll.h. Also see OEM-Configurable SDK Functionality on page 5-1.

hhpSetHardwareLineDllFileName(
PTCHAR ptcHwrFilename
)

Parameter	Description
ptcHwrFilename	Name of DLL provided by OEM containing some or all of the function exports described in the header file OemDll.h

hhpSetSymbologyDefaults

Resets an individual symbology or all symbologies to their default values.

hhpSetSymbologyDefaults(
int nSymId
)

Parameter	Description
nSymId	One of the symbology enumerated types, e.g., SYM_CODE39, SYM_OCR, or SYM_ALL to default all symbologies.

hhpUpgradeFirmware

The 5X00 Series provides the ability to update the firmware application running on the imager. hhpUpgradeFirmware checks the file contents to verify that it is a firmware application file before the file is downloaded to the imager. The firmware file is transferred to the imager compressed (lossless) unless the SDK has determined that the imager is running in bootstrap code instead of the current firmware application. In this case, the file is transferred uncompressed. This function only supports synchronous operation, so it does not return until the firmware file has been transferred to the imager and the imager has burned the new code into flash memory. When this function returns, the connection (host COM port) is connected at the default baud rate of 115200.

hhpUpgradeFirmware(
const PTCHARptcFirmwareFilename,
PDWORD DpdwTransferPercent,
HWND hTransferNotifyHwnd
)

Parameter	Description
ptcFirmwareFilename	String containing the fully qualified filename of the file that contains the code to be sent to the imager. The file extension is usually .bin or .axf. The file is sent using an Hmodem, which is a derivative of Xmodem 1K.
pdwTransferPercent	Pointer to a DWORD that contains the current percent transferred value(0 to 100). If pdwTransferPercent is valid, the transfer completion percent is written to it. This is updated after each packet is sent.
hTransferNotifyHwnd	Handle to the window that is to receive the transfer update messages. The message is sent when the percentage changes by more than 1%. The window associated with the handle should hook the WM_HHP_PROGRESS_HWND_MSG message. After file transfer is complete and while the imager is storing the new code in flash, the message WM_HHP_IMAGER_FLASHING is sent to the window HWND (if valid). The parameters are: WPARAM - Bytes transferred so far LPARAM - Bytes to be sent The window should also hook the WM_HHP_IMAGER_FLASHING message. The parameters sent are: 1st Call wParam=1 lParam=0 ransfer is done and flashing has begun Subsequent wParam=x lParam=1 Where x toggles between 0 and 1 every ½ second Final Call wParam=0 lParam=0 Flashing is complete

hhpWriteConfigItem

Writes the configuration items for one or all of the configuration structures found in the main 5X00 Series configuration structure HHP_CONFIG. Individual items can be specified by adding the appropriate mask bit by ORing it with the dwMask member of the structure. Only items whose bits are set are written; all other items are ignored.

hhpWriteConfigItem(
ConfigItems_t item,
PVOID pStruct
)

Parameter	Description	
item	One of the members of the enumerated type ConfigItems_t. The valid values are:	
	BEEPER_CONFIG	Writes the HHP_BEEPER structure settings specified by the dwMask value to the imager.
	TRIGGER_CONFIG	Writes the HHP_TRIGGER structure settings specified by the dwMask value to the imager.
	DECODER_CONFIG	Writes the HHP_DECODER_CONFIG structure settings specified by the dwMask value to the imager.
	POWER_CONFIG	Writes the HHP_POWER_SETTINGS structure settings specified by the dwMask value to the imager.
	VERSION_INFO	Returns the SDK, imager firmware, and imager boot code versions.
	SYMBOLOGY_CONFIG	Writes the HHP_SYM_CONFIG individual structure symbology settings specified by each structure's dwMask value to the imager.
	SERIAL_PORT_CONFIG	Writes the HHP_SERIAL_PORT_CONFIG structure settings specified by the dwMask value to the imager. Connection to the imager is maintaned.
	IMAGE_ACQUISITION	Writes the HHP_IMAGE_ACQUISITION structure settings specified by the dwMask value to the imager.
	IMAGE_TRANSFER	Writes the HHP_IMAGE_TRANSFER structure settings specified by the dwMask value to the imager (not progress notification methods).
	SEQUENCE_CONFIG	Returns the sequence mode enable/disable state and the data sequence command string.
	ALL_CONFIG	Writes the members specified in each of the structures of HHP_CONFIG. This includes all of the above except SERIAL_PORT_CONFIG.
pStruct	Pointer to the appropriate structure based on item: HHP_BEEPER HHP_TRIGGER HHP_DECODER_CONFIG HHP_POWER_SETTINGS HHP_SYM_CONFIG HHP_IMAGE_ACQUISITION HHP_IMAGE_TRANSFER HHP_SEQUENCE HHP_CONFIG	

hhpWriteConfigStream

Writes an entire data stream of programmable parameters to the device.

hhpWriteConfigStream(
PBYTE puchCfgStream,
int nLen
)

Parameter	Description
puchCfgStream	Buffer to hold the raw imager configuration stream.
nMaxLen	Maximum number of bytes that fit in buffer puchCfgStream.

hhpWriteSymbologyConfig

Writes configuration items for a single symbology or for all symbologies. Individual items to be written are specified by adding the appropriate mask bit (OR it) to the mask member of the structure to which it belongs. Only items whose bits are set are written; all other items are ignored.

hhpWriteSymbologyConfig(
int nSymId,
PVOID pvSymStruct
)

Parameter	Description
nSymId	One of the symbology enumerated types, e.g., SYM_CODE39, SYM_OCR, or SYM_ALL to write all symbologies.
pStruct	Pointer to the appropriate structure based on nSymbol, e.g., CODE39_T, OCR_T, or HHP_SYM_CONFIG if all symbologies.

Symbology Identifiers

Note: Please consult the appropriate symbology specification for discussion of AIM symbology IDs and modifiers.

Symbology	*Enumeration*	*AIM ID*	*Code ID*	*(hex)*
Australian Post	SYM_AUSPOST,	] X0	A	(0x41)
Aztec Code	SYM_AZTEC = 0,	] z*m*	z	(0x7A)
Aztec Mesa	SYM_MESA,	] z*m*	Z	(0x5A)
British Post	SYM_BPO,	] X0	B	(0x42)
Canadian Post	SYM_CANPOST,	] X0	C	(0x43)
China Post	SYM_CHINAPOST	] X0	Q	(0x51)
Codabar	SYM_CODABAR,	] F*m*	a	(0x61)
Codablock F	SYM_CODABLOCK,	] O*m*	q	(0x71)
Code 11	SYM_CODE11,	] H3	h	(0x68)
Code 16K	SYM_CODE16K	] K*m*	o	(0x6F)
Code 128	SYM_CODE128,	] C*m*	j	(0x6A)
Code 32 Pharmaceutical (PARAF)	SYM_CODE32,	] X0	<	(0x3C)
Code 39	SYM_CODE39,	] A*m*	b	(0x62)
Code 49	SYM_CODE49,	] T*m*	l	(0x6C)
Code 4CB (4 State Customer Barcode)	SYM_CODE4CB	] X0	M	(0x4D)
Code 93	SYM_CODE93,	] G*m*	i	(0x69)
Data Matrix	SYM_DATAMATRIX,	] d*m*	w	(0x77)
EAN-8	SYM_EAN8,	] E4	D	(0x44)
EAN-13	SYM_EAN13,	] E0	d	(0x64)
EAN•UCC Composite	SYM_COMPOSITE,	] e*m*	y	(0x79)
Interleaved 2 of 5	SYM_INT25,	] I*m*	e	(0x65)
ISBT 128	SYM_ISBT,	] C4	j	(0x6A)
Japanese Post	SYM_JAPOST,	] X0	J	(0x4A)

Symbology	*Enumeration*	*AIM ID*	*Code ID*	*(hex)*
KIX (Netherlands) Post	SYM_DUTCHPOST,	] X0	K	(0x4B)
Korea Post	SYM_KORPOST,	] X0	?	(0x3F)
Matrix 2 of 5	SYM_MATRIX25,	] X0	m	(0x6D)
MaxiCode	SYM_MAXICODE,	] U*m*	x	(0x78)
MicroPDF417	SYM_MICROPDF,	] L*m*	R	(0x52)
MSI	SYM_MSI,	] M*m*	g	(0x67)
PDF417	SYM_PDF417,	] L*m*	r	(0x72)
Planet Code	SYM_PLANET,	] X0	L	(0x4C)
Plessey Code	SYM_PLESSEY	] P0	n	(0x6E)
PosiCode	SYM_POSICODE	] p*m*	W	(0x57)
Postnet	SYM_POSTNET,	] X0	P	(0x50)
OCR US Money Font	SYM_OCR,	] o3	O	(0x4F)
QR Code	SYM_QR,	] Q*m*	s	(0x73)
Reduced Space Symbology	SYM_RSS,	] e*m*	y	(0x79)
Straight 2 of 5 Industrial	SYM_STRT25	] S0	f	(0x66)
Straight 2 of 5 IATA	SYM_IATA25,	] R*m*	f	(0x66)
TCIF Linked Code 39 (TLC39)	SYM_TLCODE39,	] L2	T	(0x54)
Telepen	SYM_TELEPEN	] B*m*	t	(0x74)
Trioptic Code	SYM_TRIOPTIC	] X0	=	(0x3D)
UPC-A	SYM_UPCA,	] E0	c	(0x63)
UPC-A with Extended Coupon Code	SYM_COUPONCODE	] E3	c	(0x63)
UPC-E	SYM_UPCE0,	] E0	E	(0x45)
UPC-E1	SYM_UPCE1,	] X0	E	(0x45)
UPU 4 State ID Tag	SYM_UPUIDTAG	] X0	N	(0x4E)

3 Enumerated Types and Definitions

Error Codes

RESULT_INTIALIZE = -1	Initial error code value.
RESULT_SUCCESS = 0	Operation was successful.
RESULT_EOT = 256	Undefined error.
RESULT_ERR_BADFILENAME	Bad file name.
RESULT_ERR_BADINTELIMAGE	Part of image window outside barcode image boundaries.
RESULT_ERR_BADPORT	Invalid connection specified.
RESULT_ERR_BADREGION	Invalid image window.
RESULT_ERR_BADSMARTIMAGE	The device did not capture a valid image for intelligent imaging.
RESULT_ERR_BAUD_TOO_HIGH	Requested baud rate not supported by host port.
RESULT_ERR_BUFFER_TOO_SMALL	Buffer passed in to small for output.
RESULT_ERR_CAPTURE_IMAGE_FAILED	Imager failed to capture the image.
RESULT_ERR_COMPRESSION_FAILED	Error compressing image data.
RESULT_ERR_CONNECT_BOOT_CODE	Imager connected but is running in boot code.
RESULT_ERR_DECOMPRESSION_FAILED	Error decompressing image data.
RESULT_ERR_DLL_FILE	Dll file specified to SetHardwareDllFileName not found.
RESULT_ERR_DRIVER	Communication error/no response.
RESULT_ERR_ENGINEBUSY	The scan engine temporarily busy.
RESULT_ERR_FILE	Error occurred during a file operation.
RESULT_ERR_FILEINCOMPATIBLE	The selected file is incompatible with the imager.
RESULT_ERR_FILEINVALID	The selected file is invalid or corrupt.
RESULT_ERR_INTERNAL_ERROR	Generic internal failure.
RESULT_ERR_INVALID_COMM_PARAMS	Invalid RS-232 parameters specified.
RESULT_ERR_MEMORY	Out of memory/memory allocation failed.
RESULT_ERR_MENUDECODE	Symbol decoded is a menu symbol.
RESULT_ERR_NAK	Received a NAK on response.
RESULT_ERR_NODECODE	No decode: timed out or no more trigger.
RESULT_ERR_NODRIVER	Communication initialization failed.
RESULT_ERR_NOIMAGE	hhpGetLastImage called but no image available.
RESULT_ERR_NOINTELBARCODE	No decode during intelligent image capture.
RESULT_ERR_NOINTELIMAGE	Error on retrieve intelligent image from imager.
RESULT_ERR_NORESPONSE	Imager did not acknowledge request.
RESULT_ERR_NOTCONNECTED	Imager not yet connected.
RESULT_ERR_NOTRIGGER	During wait for decode, checks that trigger return is released.
RESULT_ERR_PARAMETER	One of the function parameters was invalid.
RESULT_ERR_POLLEVENT	Error configuring transfer thread.
RESULT_ERR_READTHREAD_START	Error starting asynchronous transfer thread.
RESULT_ERR_READTHREAD_STOP	Error stopping asynchronous transfer thread.
RESULT_ERR_REFLASH	Engine firmware is corrupt.

Error Codes (Continued)

RESULT_ERR_SHIP_IMAGE_FAILED	Imager failed to ship captured image.
RESULT_ERR_SMARTIMAGETOOLARGE	The captured image is too large to perform intelligent imaging.
RESULT_ERR_SYMBOLOGY_HAS_NO_RANGE	The symbology has no range maximum/minimum values.
RESULT_ERR_UNICODE_UNSUPPORTED	Attempted to set Code Page, but SDK is not UNICODE.
RESULT_ERR_UNSUPPORTED	The operation was not supported by the engine.
RESULT_ERR_UPGRADE	Upgrade of imager firmware failed.
RESULT_ERR_USER_CANCEL	User called hhpCancelIo to abort operation.
RESULT_ERR_WRONGRESULTSTRUCT	Wrong structure passed in for the type specified.

Setup Type Enumerated Type

SETUP_TYPE_CURRENT,	The current value in flash.
SETUP_TYPE_DEFAULT = 0,	Hard coded Value. Set to current when imager “Reset To Defaults.”

Symbology ID Enumeration

NUM_SYMBOLOGIES	Total number of supported symbologies	All Decoders
SYM_ALL=100	All active symbologies	All Decoders
SYM_AUSPOST,	Australian Post	2D Decoder only
SYM_AZTEC = 0,	Aztec Code	2D Decoder only
SYM_BPO,	British Post	2D Decoder only
SYM_CANPOST,	Canadian Post	2D Decoder only
SYM_CHINAPOST	China Post	All Decoders
SYM_CODABAR,	Codabar	All Decoders
SYM_CODABLOCK,	Codablock	2D Decoder only
SYM_CODE11,	Code 11	All Decoders
SYM_CODE16K	Code 16K	All Decoders
SYM_CODE128,	Code 128	All Decoders
SYM_CODE32	Code 32 Pharmaceutical (PARAF)	All Decoders
SYM_CODE39,	Code 39	All Decoders
SYM_CODE49,	Code 49	All Decoders
SYM_CODE4CB	4 State Customer Barcode	2D Decoder only
SYM_CODE93,	Code 93	All Decoders
SYM_COMPOSITE,	Composite Code	2D and PDF Decoders only
SYM_COUPONCODE	UPC-A with Extended Coupon Code	All Decoders
SYM_DATAMATRIX,	Data Matrix	2D Decoder only
SYM_DUTCHPOST,	KIX (Netherlands) Post	2D Decoder only
SYM_EAN13,	EAN-13	All Decoders
SYM_EAN8,	EAN-8	All Decoders
SYM_IATA25,	Straight 2 of 5 IATA	All Decoders

Symbology ID Enumeration (Continued)

SYM_INT25,	Interleaved 2 of 5	All Decoders
SYM_ISBT,	ISBT	All Decoders
SYM_JAPOST,	Japanese Post	2D Decoder only
SYM_KORPOST	Korean Post	All Decoders
SYM_MATRIX25,	Matrix 2 of 5	All Decoders
SYM_MAXICODE,	MaxiCode	2D Decoder only
SYM_MESA,	Aztec Mesas	2D Decoder only
SYM_MICROPDF,	MicroPDF417	2D and PDF Decoders
SYM_MSI,	MSI Code	All Decoders
SYM_OCR,	OCR (OCR-A, OCR-B, OCR US Money Font, MICR)	2D Decoder only
SYM_PDF417,	PDF417	2D and PDF Decoders
SYM_PLANET,	Planet Code	2D Decoder only
SYM_PLESSEY	Plessey Code	All Decoders
SYM_POSICODE	PosiCode	All Decoders
SYM_POSTNET,	Postnet	2D Decoder only
SYM_QR,	QR Code	2D Decoder only
SYM_RSS,	Reduced Space Symbology (RSS)	All Decoders
SYM_STRT25	Straight 2 of 5 Industrial	All Decoders
SYM_TELEPEN	Telepen	All Decoders
SYM_TLCODE39,	TCIF Linked Code 39 (TLC39)	All Decoders
SYM_TRIOPTIC	Trioptic Code	All Decoders
SYM_UPCA,	UPC-A	All Decoders
SYM_UPCE0,	UPC-E	All Decoders
SYM_UPCE1,	UPC-E1 (not truly standard)	All Decoders
SYM_UPUIDTAG	ID tag (UPU 4-State)	2D Decoder only

Supported OCR Fonts

These are mutually exclusive. Only one font can be enabled at one time.

OCR_DISABLED = 0,	Disable OCR Codes.
OCR_A,	Enable OCR-A Font Decoding.
OCR_B,	Enable OCR-B Font Decoding.
OCR_MONEY,	Enable Money Font Decoding.
OCR_MICR_UNSUPPORTED,	Not supported.

Image Formats

FF_RAW_BINARY = 0,	1 bit per pixel – Each row padded out to byte boundary.
FF_RAW_GRAY,	8 bits per pixel.
FF_TIFF_BINARY,	TIFF bilevel uncompressed.
FF_TIFF_BINARY_PACKBITS,	TIFF bilevel packbits compressed.

Image Formats (Continued)

FF_TIFF_GRAY,	TIFF 8 bits per pixel uncompressed.
FF_JPEG_GRAY,	JPEG lossy compression.
FF_BMP_GRAY	Windows BMP file uncompressed.

Compression Mode Formats

COMPRESSION_NONE=0,	No compression.
COMPRESSION_LOSSLESS,	Huffman lossless compression.
COMPRESSION_LOSSY	JPEG lossy compression.

Capture Illumination Duty Cycle

HHP_DUTY_CYCLE_OFF =0,	Keep off during image capture.
HHP_DUTY_CYCLE_ON	Turn on for image capture.

#define	HHP_CAPTURE_ALWAYS_OFF	HHP_DUTY_CYCLE_OFF
#define	HHP_CAPTURE_ALWAYS_ON	HHP_DUTY_CYCLE_ON

Auto Exposure Type

HHP_AUTOEXPOSURE_BARCODE=0,	Autoexposure for decode image (darker with less noise).
HHP_AUTOEXPOSURE_PHOTO,	Autoexposure for pictures. (lighter, more pleasing image).
HHP_AUTOEXPOSURE_MANUAL	No Autoexposure. User should supply Exposure, Gain, and Frame Rate Values.

#define	HHP_AUTOEXPOSURE_NONE	HHP_AUTOEXPOSURE_MANUAL;
#define	HHP_AUTOEXPOSURE_FIXED	HHP_AUTOEXPOSURE_NONE;

Gain Values Enum

Only used when in manual capture mode.

HHP_GAIN_1x=1,
HHP_GAIN_2x,
HHP_GAIN_4x=4,

Frame Rates Enum

Only used when in manual capture mode.

HHP_1_FRAMES_PER_SEC=1,
HHP_2_FRAMES_PER_SEC,
HHP_3_FRAMES_PER_SEC,
HHP_4_FRAMES_PER_SEC,
HHP_5_FRAMES_PER_SEC,
HHP_6_FRAMES_PER_SEC,
HHP_10_FRAMES_PER_SEC=10,
HHP_12_FRAMES_PER_SEC=12,
HHP_15_FRAMES_PER_SEC=15,
HHP_20_FRAMES_PER_SEC=20,

HHP_30_FRAMES_PER_SEC=30

Beeper Volume Enum

BEEP_OFF=0,	Don't sound beeper – no volume.
BEEP_LOW=1,	Low volume.
BEEP_MEDIUM=2,	Medium volume.
BEEP_HIGH=3	Loudest volume.

Decoder Mode Enum

DECODE_METHOD_STANDARD=0,	Normal decode mode (default).
DECODE_METHOD_QUICK_OMNI	Fast omni directional decoder .
DECODE_METHOD_NONOMNI_ALD	Non-omni advanced linear decoder.
DECODE_METHOD_OMNI_LINEAR	Omni-directional linear decoder.

System (MPU) Clock Speeds

Note: CPU clock speed must always be greater than system clock speed.

POWER_SYS_SPEED_96MHZ=0,
POWER_SYS_SPEED_48MHZ,
POWER_SYS_SPEED_32MHZ

Configuration Structure Item Enum for hhpReadConfigItem() and hhpWriteConfigItem()

BEEPER_CONFIG=0,	Read/Write HHP_BEEPER items.
TRIGGER_CONFIG,	Read/Write HHP_TRIGGER_CONFIG items.
DECODER_CONFIG,	Read/Write HHP_DECODER_CONFIG items.
POWER_CONFIG,	Read/Write HHP_POWER_SETTINGS items.
VERSION_INFO,	Read/Write HHP_VERSION_INFO items.
SYMBOLOGY_CONFIG,	Read/Write HHP_SYM_CONFIG items. If specified in hhpReadConfigItem(), all the symbology config items for all the symbologies will be retrieved. Use hhpReadSymbologyConfig() to specify individual symbology structures.
SERIAL_PORT_CONFIG,	Read/Write HHP_SERIAL_PORT_CONFIG items.
IMAGE_ACQUISITION,	Read/Write HHP_IMAGE_ACQUISITION items.
IMAGE_TRANSFER,	Read/Write HHP_IMAGE_TRANSFER items.
SEQUENCE_CONFIG	Read/Write SEQUENCE_CONFIG items.
ALL_CONFIG	Read/Write HHP_CONFIG items from structure memory in all HHP configuration member structures.

Trigger Modes Enum

HHP_MANUAL_SERIAL = 0,	Requires trigger to scan (either software or hardware). No Low Power Mode.
UNUSED,	Unused.
HHP_MANUAL_LOW_POWER,	Requires trigger to scan (either software or hardware). Image enters low power mode based on low power mode timeouts.
HHP_PRESENTATION,	The imager uses ambient light to detect barcodes. For further information please see the 5X80 Integration Manual.
HHP_SCANSTAND,	Constantly looks for the scan stand barcode. When the imager detects a barcode that is different from than the scan stand barcode, it initiates a full trigger. For further information please see the 5X80 Integration Manual.
HHP_HOST_NOTIFY	
HHP_SNAP_AND_SHIP	Captures and sends an image when triggered (instead of scanning for barcodes).

Sequence Mode

HHP_SEQ_DISABLED = 0,	Sequence mode disabled.
HHP_SEQ_ENABLED,	Sequence mode enabled but not required.
HHP_SEQ_REQUIRED	Sequence mode enabled and required.

Serial Port Baud Rates

SERIAL_BAUD_300 = 300,	300 BPS
SERIAL_BAUD_600 = 600,	600 BPS
SERIAL_BAUD_1200 = 1200,	1200 BPS
SERIAL_BAUD_2400 = 2400,	2400 BPS
SERIAL_BAUD_4800 = 4800,	4800 BPS
SERIAL_BAUD_9600 = 9600,	9600 BPS
SERIAL_BAUD_19200 = 19200,	19200 BPS
SERIAL_BAUD_38400 = 38400,	38400 BPS
SERIAL_BAUD_57600 = 57600,	57600 BPS
SERIAL_BAUD_115200 = 115200,	115200 BPS
// Baud Rates that Require USB Serial or SIO950 Compatible Serial Port Driver	
SERIAL_BAUD_230400 = 230400,	230400 BPS
SERIAL_BAUD_460800 = 460800,	460800 BPS
SERIAL_BAUD_921600 = 921600,	921600 BPS

Baud Rates that Require USB Serial or SIO950 Compatible Serial Port Driver

SERIAL_BAUD_230400 = 230400,	230400 BPS
SERIAL_BAUD_460800 = 460800,	460800 BPS
SERIAL_BAUD_921600 = 921600	921600 BPS

Serial Data Bits

SERIAL_DATA_BITS_7 = 7,	7 bit data
SERIAL_DATA_BITS_8	8 bit data

Parity

SERIAL_PARITY_NONE = 'N',	No parity
SERIAL_PARITY_ODD = 'O',	Odd parity
SERIAL_PARITY_EVEN = 'E',	Even parity
SERIAL_PARITY_MARK = 'M',	Mark parity
SERIAL_PARITY_SPACE = 'S'	Space parity

Stop Bits

SERIAL_ONE_STOPBITS = 1,	1 stop bit
SERIAL_TWO_STOPBITS,	2 stop bits

Connection Types

Currently limited to serial ports.

HHP_COM1=0,	COM1
HHP_COM2,	COM2
HHP_COM3,	COM3
HHP_COM4,	COM4
HHP_COM5,	COM5
HHP_COM6,	COM6
HHP_COM7,	COM7
HHP_COM8,	COM8
HHP_COM9,	COM9
HHP_COM10,	COM10
HHP_COM11,	COM11
HHP_COM12,	COM12
HHP_COM13,	COM13
HHP_COM14,	COM14
HHP_COM15,	COM15
HHP_COM16,	COM16
HHP_COM17,	COM17
HHP_COM18,	COM18
HHP_COM19,	COM19
HHP_LAST_COMM_PORT=255	Last valid comm port.

Decoder Symbology Support

HHP_1D_CODES_ONLY=0,	1D Linear and stacked linear codes only.
HHP_1D_AND_PDF_CODES,	Same as above plus PDF417 and MicroPDF417.
HHP_1D_AND_2D_CODES	All symbologies.

HHP Action Commands

HHP_AIMER_CMD=0,	Aimer command (turn aimers on/off).
HHP_ILLUMINATION_CMD,	No longer supported by imagers.
HHP_BEEP_CMD	Beeper command (sound a single or double beep).

On/Off Enum

HHP_OFF=0,	Turn Off (HHP_AIMER_CMD and HHP_ILLUMINATION_CMD).
HHP_ON=1	Turn On (HHP_AIMER_CMD and HHP_ILLUMINATION_CMD).

Beep Execute Enum

HHP_SINGLE_BEEP=0,	Single Beep (HHP_BEEP_CMD).
HHP_DOUBLE_BEEP=1	Double Beep (HHP_BEEP_CMD).

Imager Type Enum

HHP_UNKNOWN_IMAGER	Unable to determine engine type.
HHP_DECODED_IMAGER_4080	Serial (RS-232) 4080 imager with internal decoder.
HHP_DECODED_IMAGER_4080_USB	USB serial 4080 imager with internal decoder.
HHP_DECODED_IMAGER_5080VGA	Serial (RS-232) 5080 VGA imager with internal decoder.
HHP_DECODED_IMAGER_5080VGA_USB	USB serial 5080 VGA imager with internal decoder.
HHP_DECODED_IMAGER_5080VGA_PSOC	Serial (RS-232) 5080 VGA imager with internal decoder and PSOC.
HHP_DECODED_IMAGER_5080VGA_PSOC_USB	USB serial 5080 VGA imager with internal decoder and PSOC.
HHP_DECODED_IMAGER_5080	Serial (RS-232) 5080 imager with internal decoder.
HHP_DECODED_IMAGER_5080_USB	USB serial 5080 imager with internal decoder.
HHP_DECODED_IMAGER_5080_PSOC	Serial (RS-232) 5080 imager with internal decoder and PSOC.
HHP_DECODED_IMAGER_5080_PSOC_USB	USB serial 5080 imager with internal decoder and PSOC.

Illumination Color Enum (5X10/5X80 engines only)

SECONDARY_LEDS	Alternate illumination color (if supported by engine).
PRIMARY_LEDS	Primary illumination color.

4 Structures and Mask Definitions

*Important: All structures have a dwStructSize member that **MUST BE SET** before calling any of the 5X00 Series API functions. This insures that the proper structure has been passed to the function being called.*

Most structures have a dwMask member. This specifies which structure members are active (to be read or written). Select structure items by ORing the individual structure masks for each of the items you wish to be active together.

Symbology Structures and Defines

All symbology structures are 1 of 3 possibilities.

Important: Symbology configuration definitions for each symbology are ORed together in the dwFlags member of the symbology structure. The applicable flags are dependent on symbology. For example, Aztec Code accepts only SYMBOLOGY_ENABLE, while Postnet accepts SYMBOLOGY_ENABLE and SYMBOLOGY_CHECK_TRANSMIT.

#define SYMBOLOGY_ENABLE	0x00000001	Enable Symbology bit.
#define SYMBOLOGY_CHECK_ENABLE	0x00000002	Enable usage of check character.
#define SYMBOLOGY_CHECK_TRANSMIT	0x00000004	Send check character.
#define SYMBOLOGY_START_STOP_XMIT	0x00000008	Include the start and stop characters in the decoded result string.
#define SYMBOLOGY_ENABLE_APPEND_MODE	0x00000010	Code39 append mode.
#define SYMBOLOGY_ENABLE_FULLASCII	0x00000020	Enable Code39 Full ASCII.
#define SYMBOLOGY_NUM_SYS_TRANSMIT	0x00000040	UPC-A/UPC-E Send Num Sys.
#define SYMBOLOGY_2_DIGIT_ADDENDA	0x00000080	Enable 2 digit Addenda (UPC and EAN).
#define SYMBOLOGY_5_DIGIT_ADDENDA	0x00000100	Enable 5 digit Addenda (UPC and EAN).
#define SYMBOLOGY_ADDENDA_REQUIRED	0x00000200	Only allow codes with addenda (UPC and EAN).
#define SYMBOLOGY_ADDENDA_SEPARATOR	0x00000400	Include Addenda separator space in returned string.
#define SYMBOLOGY_EXPANDED_UPCE	0x00000800	Extended UPC-E.
#define SYMBOLOGY_UPCE1_ENABLE	0x00001000	UPC-E1 enable (use SYMBOLOGY_ENABLE for UPC-E0).
#define SYMBOLOGY_COMPOSITE_UPC	0x00002000	Enable UPC Composite codes.
#define SYMBOLOGY_AZTEC_RUNE	0x00004000	Enable Aztec Code Run.
#define SYMBOLOGY_AUSTRALIAN_BAR_WIDTH	0x00010000	Include Australian postal bar data in string.
#define SYMBOLOGY_AUS_CUST_FIELD_NUM	0x00020000	Customer fields as numeric data using the N Table.
#define SYMBOLOGY_AUS_CUST_FIELD_AlPHA	0x00040000	Customer fields as alphanumeric data using the C Table.

Flags for Aztec Mesas are reused, since Aztec Mesas have no addenda, extended UPC-E, or UPC-E1 enable flags.

#define SYMBOLOGY_ENABLE_MESA_IMS	0x00020000	Mesa IMS enable.
#define SYMBOLOGY_ENABLE_MESA_1MS	0x00040000	Mesa 1MS enable.
#define SYMBOLOGY_ENABLE_MESA_3MS	0x00080000	Mesa 3MS enable.
#define SYMBOLOGY_ENABLE_MESA_9MS	0x00100000	Mesa 9MS enable.
#define SYMBOLOGY_ENABLE_MESA_UMS	0x00200000	Mesa UMS enable.
#define SYMBOLOGY_ENABLE_MESA_EMS	0x00400000	Mesa EMS enable.
#define SYMBOLOGY_ENABLE_MESA_MASK	0x007E0000	Enable all Mesa.

There is only one symbology ID for RSE, RSL, and RSS, so 3 flags are used for enable.

#define SYMBOLOGY_RSE_ENABLE	0x00800000	Enable RSE Symbology bit.
#define SYMBOLOGY_RSL_ENABLE	0x01000000	Enable RSL Symbology bit.
#define SYMBOLOGY_RSS_ENABLE	0x02000000	Enable RSS Symbology bit.
#define SYMBOLOGY_RSX_ENABLE_MASK	0x03800000	RSS enable all.

Telepen and PosiCode

#define SYMBOLOGY_TELEPEN_OLD_STYLE	0x04000000	Telepen Old Style mode.
#define SYMBOLOGY_POSICODE_LIMITED_1	0x08000000	PosiCode Limited of 1

#define SYMBOLOGY_POSICODE_LIMITED_2	0x10000000	PosiCode Limited of 2
#define SYMBOLOGY_CODABAR_CONCATENATE	0x20000000	Codabar concatenate.

Flags for OCR are reused, since none of the other flags apply to OCR.

#define SYMBOLOGY_ENABLE_OCR_A	0x00000001	OCR-A enable.
#define SYMBOLOGY_ENABLE_OCR_B	0x00000002	OCR-B enable.
#define SYMBOLOGY_ENABLE_OCR_MONEY	0x00000004	OCR-Money enable.
#define SYMBOLOGY_ENABLE_OCR_MICR	0x00000008	OCR-Micr enable.

Symbology structure sets masks to specify which items of config structure are to be set or read.

#define SYM_MASK_FLAGS	0x00000001	Flags are valid.
#define SYM_MASK_MIN_LEN	0x00000002	Min Length valid.
#define SYM_MASK_MAX_LEN	0x00000004	Max Length valid.
#define SYM_MASK_OCR_MODE	0x00000008	OCR mode valid.
#define SYM_MASK_DIRECTION	0x00000010	OCR direction valid.
#define SYM_MASK_TEMPLATE	0x00000020	OCR template valid.
#define SYM_MASK_GROUP_H	0x00000040	OCR group H valid.
#define SYM_MASK_GROUP_G	0x00000080	OCR group H valid.
#define SYM_MASK_CHECK_CHAR	0x00000100	OCR check char valid.
#define SYM_MASK_ALL	0xffffffff	Generic all mask.

Structure for symbologies with no specified minimum or maximum length:

```
typedef struct _tagSymFlagsOnly
{
        DWORD       dwStructSize;   Set to sizeof( SymFlagsOnly_t );
        DWORD       dwMask;         Mask which can only be 0 or SYM_MASK_FLAGS.
        DWORD       dwFlags;        OR of valid flags for the given symbology.

} SymFlagsOnly_t, *PSymFlagsOnly_t;
```

Min/Max barcode lengths for symbologies that have length settings:

Code	Min	Max
Aztec	1	3750
China Post	2	80
Codabar	2	60
Codablock F	1	2048
Code 11	1	80
Code 16K	1	160
Code 128	0	80
Code 2 of 5	1	48
Code 39	0	48
Code 49	1	81
Code 93	0	80
DataMatrix	1	1500
EAN•UCC Composite	1	2435
IATA Code 2 of 5	1	48
Interleaved 2 of 5	2	80
Korean Post	2	80
Matrix 2 of 5	1	80
MaxiCode	1	150
MicroPDF417	1	366
MSI	4	48
PDF417	1	2750
Plessey Code	4	48
PosiCode	2	80
QR Code	1	3500
Reduced Space Symbology (RSS)	4	74
Telepen	1	60

Structure for symbologies with minimum and maximum length:

```
typedef struct _tagSymFlagsRange
{
        DWORD       dwStructSize;       Set to sizeof( SymFlagsRange_t );
        DWORD       dwMask;             Item Masks – SYM_MASK_FLAGS,MIN_LEN,MAX_LEN.
        DWORD       dwFlags;            OR of valid flags for the given symbology.
        DWORD       dwMinLen;           Minimum length for valid barcode string for this symbology.
        DWORD       dwMaxLen;           Maximum length for valid barcode string for this symbology.

} SymFlagsRange_t, *PSymFlagsRange_t;
```

Structure for unusual OCR:

```
#define MAX_TEMPLATE_LEN                256
#define MAX_GROUP_H_LEN                 256
#define MAX_GROUP_G_LEN                 256
#define MAX_CHECK_CHAR_LEN              64
typedef struct _tagSymCodeOCR
{
        DWORD           dwStructSize;                       Set to size of ( SymCodeOcr_t );
        DWORD           dwMask;                             Item masks.
        OCRMode_t       ocrMode;                            OCR Enable/Mode structure.
        OCRDirection_t  ocrDirection;                       OCR direction structure.  Not supported.
        TCHAR           tcTemplate[ MAX_TEMPLATE_LEN ];     Template for decoded data ('d' - decimal,
                                                            'a' - ASCII, 'l' - letter, 'e' - extended).
        TCHAR           tcGroupG[ MAX_GROUP_H_LEN ];        Group G character string.
```

TCHAR	tcGroupH[MAX_GROUP_G_LEN];	Group H character string.
TCHAR	tcCheckChar[MAX_CHECK_CHAR_LEN];	Check character string.

} SymCodeOCR_t, *PSymCodeOCR_t;

Structure of all Symbology Structures

There is one structure for each symbology. This info is stored in imager config.

Define aliases for each symbology structure:

#define AZTEC_T	SymFlagsRange_t	Aztec Code has max and min length values.
#define MESA_T	SymFlagsOnly_t	Aztec Mesa has flags only.
#define CODABAR_T	SymFlagsRange_t	Codabar has max and min length values.
#define CODE11_T	SymFlagsRange_t	Code 11 has max and min length values.
#define CODE128_T	SymFlagsRange_t	Code 128 has max and min length values.
#define CODE39_T	SymFlagsRange_t	Code 39 has max and min length values.
#define CODE49_T	SymFlagsRange_t	Code 49 has max and min length values.
#define CODE93_T	SymFlagsRange_t	Code 93 has max and min length values.
#define COMPOSITE_T	SymFlagsRange_t	Composite code has max and min length values.
#define DATAMATRIX_T	SymFlagsRange_t	Data Matrix has max and min length values.
#define EAN8_T	SymFlagsOnly_t	EAN-8 has flags only.
#define EAN13_T	SymFlagsOnly_t	EAN-13 has flags only.
#define INT25_T	SymFlagsRange_t	Interleaved 2 of 5 has max and min length values.
#define MAXICODE_T	SymFlagsRange_t	MaxiCode has max and min length values.
#define MICROPDF_T	SymFlagsRange_t	MicroPDF417 has max and min length values.
#define OCR_T	SymCodeOCR_t	OCR has its own structure.
#define PDF417_T	SymFlagsRange_t	PDF417 has max and min length values.
#define POSTNET_T	SymFlagsOnly_t	Postnet has flags only.
#define QR_T	SymFlagsRange_t	QR Code has max and min length values.
#define RSS_T	SymFlagsRange_t	RSS Codes have max and min length values.
#define UPCA_T	SymFlagsOnly_t	UPC-A has flags only.
#define UPCE_T	SymFlagsOnly_t	UPC-E has flags only.
#define ISBT_T	SymFlagsOnly_t	ISBT Code has flags only.
#define BPO_T	SymFlagsOnly_t	British Post has flags only.
#define CANPOST_T	SymFlagsOnly_t	Canadian Post has flags only.
#define AUSPOST_T	SymFlagsOnly_t	Australian Post has flags only.
#define IATA25_T	SymFlagsRange_t	IATA 2 of 5 has max and min length values.
#define CODABLOCK_T	SymFlagsRange_t	Codablock has max and min length values.
#define JAPOST_T	SymFlagsOnly_t	Japanese Post has flags only.
#define PLANET_T	SymFlagsOnly_t	Planet Code has flags only.
#define DUTCHPOST_T	SymFlagsOnly_t	KIX Post has flags only.
#define MSI_T	SymFlagsRange_t	MSI Code has max and min length values.
#define TLCODE39_T	SymFlagsOnly_t	TLCode 39 has flags only.
#define MATRIX25_T	SymFlagsRange_t	Matrix 2 of 5 Code has max & min length values.
#define KORPOST_T	SymFlagsOnly_t	Korea Post only has enable.
#define TRIOPTIC_T	SymFlagsOnly_t	Trioptic Code has flags only.
#define CODE32_T	SymFlagsOnly_t	Code 32 has flags only.
#define CODE25_T	SymFlagsRange_t	Code 2 of 5 has min and max length values.
#define PLESSEY_T	SymFlagsRange_t	Plessey Code has min and max length values.
#define CHINAPOST_T	SymFlagsRange_t	China Post has min and max length values.
#define TELEPEN_T	SymFlagsRange_t	Telepen Code has min and max length values.
#define CODE16K_T	SymFlagsRange_t	Code 16K has min and max length values.
#define POSICODE_T	SymFlagsRange_t	PosiCode has min and max length values.
#define COUPONCODE_T	SymFlagsOnly_t	UPC Coupon code has flags only.
#define CODE4CB_T	SymFlagsOnly_t	4 State Customer Barcode.
#define UPUIDTAG_T	SymFlagsOnly_t	ID tag (UPU 4-State) has flags only.

typedef struct _tagSymCfg
{

DWORD	dwStructSize;	Set to sizeof(SymCfg_t);

Linear Codes - Flags supported for this code:

CODABAR_T	codabar;	Enable,Check,CheckSend,StartStop,Concatenate
CODE32_T	code32;	Enable
COUPONCODE_T	couponCode;	Enable
TRIOPTIC_T	triopticCode;	Enable
CODE11_T	code11;	Enable,Check,CheckSend
CODE128_T	code128;	Enable
CODE39_T	code39;	Enable,Check,CheckSend,StartStop,Append,FullAscii
CODE49_T	code49;	Enable
CODE93_T	code93;	Enable
COMPOSITE_T	composite;	Enable,CompositeUPC
EAN8_T	ean8;	Enable, Check, Addenda2, Addenda5, AddendaReq, AddendaSep
EAN13_T	ean13;	Enable, Check, Addenda2, Addenda5, AddendaReq, AddendaSep
IATA25_T	iata25;	Enable
INT25_T	int2of5;	Enable,Check,CheckSend
ISBT_T	isbt;	Enable
MATRIX25_T	matrix25;	Enable
MSI_T	msi;	Enable,Check
UPCA_T	upcA;	Enable, check, NumSysTrans, Addenda2, Addenda5, AddendaReq, AddendaSep
UPCE_T	upcE;	Enable, Check, NumSysTrans, Addenda2, Addenda5, AddendaReq, AddendaSep, ExpandedE, EnableE1

Postal Codes

AUSPOST_T	australiaPost;	Enable,AustralianBar
BPO_T	britishPost;	Enable
CANPOST_T	canadaPost;	Enable
DUTCHPOST_T	dutchPost;	Enable
JAPOST_T	japanPost;	Enable
KORPOST_T	koreaPost	Enable
PLANET_T	usPlanet;	Enable,Check
POSTNET_T	usPostnet;	Enable,Check
CHINAPOST_T	chinaPost;	Enable
UPUIDTAG_T	upuIdTag;	Enable
CODE4CB_T	code4CB;	Enable

2D Codes

AZTEC_T	aztec;	Enable,AztecRun
MESA_T	aztecMesa;	EnableIMS, Enable1MS, Enable3MS, Enable9MS, EnableUMS, EnableEMS
CODABLOCK_T	codablock;	Enable
DATAMATRIX_T	datamatrix;	Enable
MAXICODE_T	maxicode;	Enable,SCMOnly
MICROPDF_T	microPDF;	Enable
PDF417_T	pdf417;	Enable
QR_T	qr;	Enable
RSS_T	rss;	Enable
TLCODE39_T	tlCode39;	Enable
CODE25_T	code2of5;	Enable
PLESSEY_T	plesseyCode;	Enable
TELEPEN_T	telepen;	Enable,Old Style Mode
CODE16K_T	code16k;	Enable

```
        POSICODE_T          posiCode;           Enable,Limited 1, Limited 2
        Special OCR "code"
        OCR_T               ocr;                None (See SymCodeOCR_t)

} SymCfg_t, HHP_SYM_CONFIG, *PSymCfg_t, *PHHP_SYM_CONFIG;
```

Data structures for decoded barcode message: hhpCaptureBarcode() and hhpGetAsyncResults(). Not stored in imager.

```
  #define MAX_MESAGE_LENGTH       4096

typedef struct _tagHHP_DECODE_MSG
{
        DWORD       dwStructSize;                       Size of decode structure.
        TCHAR       pchMessage[ MAX_MESAGE_LENGTH ];    decoded message data
        TCHAR       chCodeID;                           AIM Id of symbology
        TCHAR       chSymLetter;                        Hand Held Products Id of symbology
        TCHAR       chSymModifier;                      Modifier characters
        DWORD       nLength;                            length of the decoded message

} HHP_DECODE_MSG, DecodeMsg_t, *PHHP_DECODE_MSG;

typedef struct _tagHHP_RAW_DECODE_MSG
{
        WORD        wStructSize;                        Size of decode structure
        BYTE        chMessage[ MAX_MESAGE_LENGTH ];     Decoded message data
        BYTE        hCodeID;                            AIM ID of symbology
        BYTE        hSymLetter;                         HHP ID of symbology
        BYTE        hSymModifier;                       Modifier characters
        DWORD       Length;                             Length of the decoded message

} HHP_RAW_DECODE_MSG, RawDecodeMsg_t, *PHHP_RAW_DECODE_MSG;
```

Imaging Structures and Defines

Image Acquisition bit mask:

The following are also configuration items:

```
#define WHITE_VALUE_MASK                    0x00001     Target value (0-255) for the "white" pixel.
#define WHITE_WINDOW_MASK                   0x00002     Acceptable delta from target white.
#define MAX_CAPTURE_RETRIES_MASK            0x00004     Max # of frames to try to get white value.
#define ILLUMINATION_DUTY_CYCLE_MASK        0x00008     How LEDs behave during image capture.
#define LIGHTS_DUTY_CYCLE_MASK              0x00008     Duplicate of above mask.
#define AIMER_DUTY_CYCLE_MASK               0x00010     How aimers behave during image capture.
#define FIXED_GAIN_MASK                     0x00020     If manual mode, gain value to use.
#define FIXED_EXPOSURE_MASK                 0x00040     If manual mode, exposure value to use.
#define FRAME_RATE_MASK                     0x00080     If manual mode, frame rate to use.
#define AUTOEXPOSURE_MODE_MASK              0x00100     Barcode, Photo, or manual AGC mode.
#define IMAGE_CAPTURE_MODE_MASK             0x00100     Same as above mask.
```

Following 2 items are not config items and are only used in hhpAcquireImage():

```
#define WAIT_FOR_TRIGGER_MASK               0x00200     Wait for trigger before capture.
#define PREVIEW_MODE_IMAGE_MASK             0x00400     Capture a preview image (214x160x8 JPEG).
```

Grouped masks:

```
#define CAPTURE_MASK_CONFIG_ALL             0x001ff     Mask for all configuration items.
#define CAPTURE_MASK_FIXED_AGC              0x00180     Mask for all manual exposure mode items.
#define CAPTURE_MASK_ALL                    0x007ff     Mask for all structure members.
```

Image Acquisition structure:

typedef struct _tagHHP_IMAGE_ACQUISITION
{

DWORD	dwStructSize;	Size of structure in bytes.
DWORD	dwMask;	Mask of active items.
Config items		
DWORD	dwWhiteValue;	Target “white pixel” value.
DWORD	dwWhiteWindow;	Acceptable delta from white value.
DWORD	dwMaxNumExposures;	Max frame capture tries for white value.
HHP_DUTY_CYCLE	illuminatCycle;	Illumination duty cycle (never on, on during imaging).
HHP_DUTY_CYCLE	aimerCycle;	Aimer duty cycle (never on, on during imaging).
HHP_GAIN	fixedGain;	If manual capture mode, gain value for capture.
DWORD	dwFixedExposure;	If manual capture mode, exposure time for capture.
HHP_FRAME_RATE	frameRate;	If manual capture mode, frame rate for capture.
HHP_AUTOEXPOSURE	captureMode;	Autoexposure (AGC) Capture mode: barcode, photo or manual.
Capture time only, not real config items:		
BOOL	bWaitForTrigger;	Wait for hardware or software trigger before capturing image.
BOOL	bPreviewImage;	Capture a preview image. These are subsample 3, full window, JPEG transfer images.

} HHP_IMAGE_ACQUISITION, *PHHP_IMAGE_ACQUISITION;

Image Transfer bit masks:

#define BITS_PER_PIXEL_MASK	0x00001	Number of bits per pixel transferred (1 or 8).
#define SUBSAMPLE_VALUE	0x00002	Subsample value (1-10).
#define SUBSAMPLE_VALUE_MASK	0x00002	Subsample value (1-10).
#define BOUNDING_RECTANGLE_MASK	0x00004	Rectangular region within image to send.
#define COMPRESSION_MODE_MASK	0x00008	No Compression, Lossless or Lossy.
#define HISTOGRAM_STRETCH_MASK	0x00010	Range -> 0 - 255.
#define COMPRESSION_FACTOR_MASK	0x00020	If lossy compression, image quality percent.
#define EDGE_ENHANCEMENT_MASK	0x00100	Edge sharpening filter.
#define GAMMA_CORRECTION_MASK	0x00200	Gamma correction.
#define TEXT_ENHANCEMENT_MASK	0x00400	Text sharpening filter.
#define INFINITY_FILTER_MASK	0x00800	Sharpening for image beyond normal focus.
#define FLIP_IMAGE_MASK	0x01000	Rotate the image 180 degrees.
#define NOISE_FILTER_MASK	0x02000	Smoothing (fly spec) filter.
#define TRANSFER_UPDATE_HWND	0x00040	Transfer progress message window.
#define TRANSFER_UPDATE_DWORD	0x00080	Pointer to DWORD for percent of transfer complete.
#define TRANSFER_MASK_ALL	0x03fff	Mask to select all configuration items in structure.
#define TRANSFER_MASK_ALL_NO_NOTIFY	0x03f3f	Mask to select all structure members.

Image Transfer structure:

typedef struct _tagHHP_IMAGE_TRANSFER
{

DWORD	dwStructSize;	Size of structure in bytes.
DWORD	dwMask;	Mask of active items.
Config items:		
DWORD	dwBitsPerPixel;	Bits per pixel for transferred image (1 or 8 bits only).
DWORD	dwSubSample;	Subsample value. This means take every dwSubSample pixels of every dwSubSample row. The default is 1.

RECT	boundingRect;	Rectangle describing a window within the image. All pixels outside this rectangle are omitted from the transferred image.
BOOL	bHistogramStretch;	Scaled frequency of total image pixels.
Compression_t	compressionMode;	How image is compressed on transfer. Compression reduces the amount of data to transfer but can reduce image quality (Lossy compression) and does take a finite length of time.
DWORD	dwCompressionFactor;	Lossy compression is JPEG lossy. If lossy compression, this value specifies the image quality percentage from 100 (virtually no loss) to 1 (very poor). Image size drops with decrease in image quality.
DWORD	dwEdgeEhancement;	A sharpening filter used to sharpen light/dark edges within the image. The valid range of values is 0 (no edge enhancement) to 23 (maximum edge enhancement).
DWORD	dwGammaCorrection;	Applies gamma correction to the image. The valid range is 0 (no gamma correction) to 1000 (maximum correction).
DWORD	dwTextEnhancement;	This filter is an edge sharpening filter optimized for text. The valid range is 0 (no text enhancement) to 255 (maximum enhancement).
BOOL	bInfinityFilter;	This is a boolean flag (TRUE or FALSE) that applies a filter to the image that sharpens objects beyond the normal focal distance of the imager.
BOOL	bFlipImage;	This is a boolean flag (TRUE or FALSE) that flips the image 180°
BOOL	bNoiseFilter;	This is a boolean flag (TRUE or FALSE) that enables or disables the imager smoothing filter.
Transfer time only, not stored in imager:		
HWND	hTransferNotifyHwnd;	The user-defined window message. WM_HHP_PROGRESS_HWND_MSG (wParam is bytes so far, lParam is bytes to send) will be sent if this member mask specified and its value is a valid windows handle.
PDWORD	pdwTransferPercent;	If non-NULL and specified in MASK, the percent complete of the transfer is placed here. It is up to the caller to check the value in a thread or timer callback.

} HHP_IMAGE_TRANSFER, *PHHP_IMAGE_TRANSFER;

Data structure for captured image: hhpAcquireImage(), hhpGetLastImage() and hhpGetAsyncResult(). Not stored in imager.

typedef struct _tagHHP_IMAGE
{

DWORD	dwStructSize;	Size of structure in bytes.
PBYTE	puchBuffer;	Buffer for image.
LONG	nBufferSize;	Size of buffer in bytes.
FileFormat_t	fileFormat;	Format for returned data.
DWORD	dwJpegQFactor;	JPEG Quality Factor.
LONG	nBytesReturned;	Number of bytes returned.
SIZE	imgSize;	Size of image returned.
LONG	nCapturedFrames;	Number of frames captured prior to this image.
LONG	nGain;	Gain value used to capture this image.
LONG	nExposureTime;	Exposure time used to capture this image.

```
    LONG        nUnderexposedPixels;    Number of underexposed pixels in image.
    LONG        nOverexposedPixels;     Number of overexposed pixels in image.

} HHP_IMAGE, *PHHP_IMAGE;
```

Other Imager Configuration Structures and Defines

Beeper member valid bitmasks:

```
#define BPMASK_ON_DECODE          0x00001   Beep on successful decode.
#define BPMASK_SHORT_BEEP         0x00002   Beep on imager reset.
#define BPMASK_MENU_CMD_BEEP      0x00004   Beep on receive menu command.
#define BPMASK_BEEP_VOLUME        0x00008   Set the beeper volume.
#define BPMASK_ALL                0x0000f   Mask for all members valid.
```

Beeper structure:

```
typedef struct _tagHHP_BEEPER
{
    DWORD           dwStructSize;   Size of structure in bytes.
    DWORD           dwMask;         Mask of active items.
    BOOL            bBeepOnDecode;  Sound beeper on successful decode.
    BOOL            bShortBeep;     Sound beeper whenever imager resets.
    BOOL            bMenuCmdBeep;   Sound beeper whenever a menu command is received.
    BEEPER_VOLUME   beepVolume;     Set the beeper volume.

} HHP_BEEPER, *PHHP_BEEPER;
```

Trigger timeout range:

```
#define MIN_TRIGGER_TIMEOUT_VAL   0         Infinite
#define MAX_TRIGGER_TIMEOUT_VAL   300000    5 Minutes
```

Trigger structure masks:

```
#define TRGMASK_TRIG_MODE   0x00001   Enable/disable manual trigger mode.
#define TRGMASK_TIMEOUT     0x00002   Sanity timeout on trigger event.
#define TRGMASK_ALL         0x00003   All members valid mask.
```

Triggering control:

```
typedef struct _tagHHP_TRIGGER
{
    DWORD           dwStructSize;   Size of structure in bytes.
    DWORD           dwMask;         Mask of active items.
    HHP_TRIG_MODES  TriggerMode;    Trigger mode.
    DWORD           dwTriggerTimeout;   0->9999 (milliseconds).

} HHP_TRIGGER, *PHHP_TRIGGER;
```

Engine information structure defines:

```
#define MAX_SHORT_VERSION_LEN   32
#define MAX_SERIAL_NUMBER_LEN   16
#define MAX_CHECKSUM_LEN        8
#define ENGINE_ID_DIGITS        4
```

Engine information structure

```
typedef struct _tagHHP_ENGINE_INFO
{
    DWORD   dwStructSize;                               Structure size
    TCHAR   tcEngId[ ENGINE_ID_DIGITS ];                4 digit ASCII Hex.
    TCHAR   tcHHPSerialNumber[ MAX_SERIAL_NUMBER_LEN ]; ASCII Decimal
    TCHAR   tcCustomSerialNumber[ MAX_SERIAL_NUMBER_LEN ]; ASCII Decimal
    LONG    nAimerX;                                    Aimer X
```

```
    LONG        nAimerY;                                    Aimer Y
    LONG        nLaserPower;                                Laser power in mW
    TCHAR       tcFirmwareChecksum[ MAX_CHECKSUM_LEN ];     Firmware Checksum (ASCII Hex)
    TCHAR       tcFirmwareRev[ MAX_SHORT_VERSION_LEN ];     Firmware revision number.Number
    LONG        nLedCtrlMode;                               How LEDs are controled.
    LONG        nLedClr;                                    LED color (red or green LEDs)
    LONG        nPwmFreq;                                   PWM base frequence.
    LONG        nRedLedCurrent;                             Red LED current (mA).
    LONG        nRedLedMaxCurrent;                          Red LED max current (mA).
    LONG        nGreenLedCurrent;                           Green LED current (mA).
    LONG        nGreenLedMaxCurrent;                        Green LED max current (mA).
    LONG        nAimerCurrent;                              Aimer current (mA).
    LONG        nAimerMaxCurrent;                           Aimer max current (mA).
    LONG        nPixelClockFreq;                            Pixel clock frequency (MHz)
    TCHAR       tcRegisterChecksum[ MAX_CHECKSUM_LEN ];     Register chcksum (ASCII hex)

} HHP_ENGINE_INFO, *PHHP_ENGINE_INFO;
```

Sequence structure defines:

```
#define MAX_SEQ_BARCODES            12
#define MAX_NUM_START_CHARS         32
#define SEQ_ALL_LENGTH              9999
```

Sequence structure masks:

```
#define SEQMASK_MODE                0x00001     sequenceMode
#define SEQMASK_BARCODES            0x00002     dwNumBarCodes & seqBarCodes[]
#define SEQMASK_ALL                 0x00003     Everything
```

Individual sequence barcode structure:

```
typedef struct _tagSeqItem
{
    LONG        nSymId;                                 Symbology identifier SYM_xxxx
    LONG        nLength;                                Match length or 9999 to match any length.
    TCHAR       tcStartChars[ MAX_NUM_START_CHARS+1 ];  Matching string (from start)

} SeqBarCode_t, *PSeqBarCode_t;
```

Sequence structure:

```
typedef struct _tagHHP_SEQUENCE_MODE
{
    DWORD           dwStructSize;                       Size of structure in bytes
    DWORD           dwMask;                             Mask of active items
    HHP_SEQ_MODES   sequenceMode;                       Disabled/Enabled/Enabled & Required
    DWORD           dwNumBarCodes;                      This MUST be sent if sending seqBarCodes
    SeqBarCode_t    seqBarCodes[ MAX_SEQ_BARCODES ];    Barcodes to sequence in order they are to be
                                                        sent
    TCHAR           tchSeqCmdBlk[ SIZE_OF_SEQUENCE_BLOCK ];

} HHP_SEQUENCE_MODE, *PHHP_SEQUENCE_MODE;
```

Decode method stucture masks:

```
#define DCMASK_MAX_MESSAGE_LENGTH   0x00001     Maximum length of decoded string. This item is Read Only.
#define DCMASK_DECODE_MULTIPLE      0x00002     Look for and report all barcodes in captured frame.
#define DCMASK_USE_AIMERS           0x00004     Use aimers when capturing barcodes.
#define DCMASK_PRINT_WEIGHT         0x00008     Relative contrast between barcode and background (0-9).
#define DCMASK_DECODE_METHOD        0x00010     Normal, linear codes only.  Fastest (may miss codes at
                                                edges of image).
#define DCMASK_CENTER_ENABLE        0x00020     Only accept barcodes whose boundaries intersect center
                                                window.
#define DCMASK_CENTER_WINDOW        0x00040     Rectangle about center of image for previous mask.
```

```
#define DCMASK_ILLUMINAT_LED_COLOR    0x00080    Illumination LED color to use.
#define DCMASK_ALL                    0x000ff    All structure members are active.
```

Decoder functionality settings:

```
typedef struct _tagHHP_DECODER_CONFIG
{
        DWORD           dwStructSize;          Size of structure in bytes.
        DWORD           dwMask;                Mask of active items.
        DWORD           dwMaxMsgSize;          Maximum length for any returned barcode string.  This is a
                                               read only value.
        BOOL            bDecodeMultiple;       Decode and send all symbols decoded with first frame
                                               where at least 1 symbol is found.
        BOOL            bUseAimers;            Turn on aimers during barcode capture.
        DWORD           dwPrintWeight;         How dark the barcode elements are relative to the back-
                                               ground (1-7).
        DECODE_METHOD   decodeMethod;          Normal decoder, linear codes only, fast normal decoder,
                                               which omits checking at the image margins as well as
                                               some bad barcode correction.
        BOOL            bCenterDecodeEnable;   Does symbol have to intersect center decode window to be
                                               valid.
        RECT            centerWindow;          Bounding coords of center window that decoded symbol
                                               must intersect
        ILLUM_LED_COLOR illumLedColor;         Illumination LED color to use.

} HHP_DECODER_CONFIG, *PHHP_DECODER_CONFIG;
```

Aimer Modes:

```
typedef enum
{
        AIMER_MODE_ALWAYS_OFF=0,       No aimer LEDs.
        AIMER_MODE_ILLUM_AND_AIM,      Aimer LEDs and illumination LEDs on simultaneously.
        AIMER_MODE_ALWAYS_ON           Aimer LEDs always on.

} HHP_AIMER_MODES;
```

Power setting item masks:

```
#define PWRMASK_TRIGGER_MODE                    0x00001
#define PWRMASK_SERIAL_TRIGGER_TIMEOUT          0x00002
#define PWRMASK_LOW_POWER_TIMEOUT               0x00004
#define PWRMASK_STOP_MODE_TIMEOUT               0x00004
#define PWRMASK_ILLUM_LED_INTENSITY             0x00008
#define PWRMASK_SYS_SPEED                       0x00010
#define PWRMASK_AIMER_MODE                      0x00020
#define PWRMASK_AIMER_DURATION                  0x00040
#define PWRMASK_AIMER_DELAY                     0x00080
#define PWRMASK_IMAGER_IDLE_TIMEOUT             0x00100
#define PWRMASK_ RS232_LOW_POWER_TIMEOUT        0x00200
#define PWRMASK_ SLEEP_MODE_TIMEOUT             0x00200
#define PWRMASK_POWER_OFF_HANDLE                0x00400
#define PWRMASK_POWER_OFF_HWND                  0x00800
#define PWRMASK_ALL                             0x00FFF
```

Matrix products power management structure:

```
typedef struct _tagHHP_POWER_SETTINGS
{
        DWORD           dwStructSize;
        DWORD           dwMask;
        HHP_TRIG_MODES  TriggerMode;
```

DWORD	dwTriggerTimeout;	0->300000 (milliseconds).
DWORD	dwLowPowerTimeout;	0 -> 300.
DWORD	dwLEDIntensityPercent;	0 -> 100%.
HHP_SYS_SPEED	systemClockSpeed;	Clock speed for reset of system (except RS-232).
HHP_AIMER_MODES	AimerMode;	Aimer always on, alternating between illumination LEDs, or disabled.
DWORD	dwAimerDuration;	0 -> 240000 (milliseconds).
DWORD	dwAimerDelay;	0 -> 240000 (milliseconds).
DWORD	dwImagerIdleTimeout;	0 -> 999999 (milliseconds).
DWORD	dw RS232LowPwrTimeout;	0 -> 300 (seconds). RS-232 inactivity timeout used to enter sleep mode.

These are used to notify on suspend (WinCE Suspend) – Not stored in imager:

HANDLE	hPowerOffHandle;	Handle for system suspend wakeup notification.
HWND	hPowerOffHwnd;	Window HWND for system suspend wakeup notification message.

} HHP_POWER_SETTINGS, *PHHP_POWER_SETTINGS;

Version setting item masks – Items are read only:

#define VERMASK_SDK_API	0x00001	SDK version number.
#define VERMASK_IMAGER_FIRMWARE	0x00002	Imager firmware version.
#define VERMASK_IMAGER_PART_NUM	0x00004	Imager part number.
#define VERMASK_IMAGER_BOOT_CODE	0x00008	Imager boot code version.
#define VERMASK_IMAGER_DEVICE_TYPE	0x00010	Device type of which imager is part.
#define VERMASK_ALL	0x0001f	All version info mask.

Revision information:

#define MAX_VERSION_STRING_LEN64
#define MAX_DEVICE_TYPE_STRING_LEN16
typedef struct _tagHHP_VERSION_INFO
{

DWORD	dwStructSize;	Size of structure in bytes.
DWORD	dwMask;	Mask of active items.
TCHAR	tcAPIRev[MAX_VERSION_STRING_LEN];	SDK API version string.
TCHAR	tcFirmwareRev[MAX_VERSION_STRING_LEN];	Imager firmware version.
TCHAR	tcPartNumber[MAX_VERSION_STRING_LEN]	;Imager firmware part number.
TCHAR	tcBootCodeRev[MAX_VERSION_STRING_LEN]	;Imager boot code version.
TCHAR	tcDeviceType[MAX_DEVICE_TYPE_STRING_LEN]	;Imager device identifier.

} HHP_VERSION_INFO, *PHHP_VERSION_INFO;

Structure that includes all structures used to configure imager. See ConfigItems_t when specifying structure to call hhpReadConfigItem() and hhpWriteConfigItem().

typedef struct _tagHHP_CONFIG
{

DWORD	dwStructSize;	Size of HHP_CONFIG structure in bytes.
HHP_BEEPER	beeperCfg;	Beeper configuration structure.
HHP_TRIGGER	triggerCfg;	Trigger configuration structure.
HHP_DECODER_CONFIG	decoderCfg;	Decoder function configuration structure.
HHP_POWER_SETTINGS	powerCfg;	Power modes configuration structure.
HHP_VERSION_INFO	versionInfo;	Version information structure.
HHP_SYM_CONFIG	symbolCfg;	Decoder symbology enables and configuration structure.
HHP_IMAGE_ACQUISITION	imgAcqu;	Image acquisition configuration structure.
HHP_IMAGE_TRANSFER	imgTrans;	Image transfer configuration structure.

} HHP_CONFIG, *PHHP_CONFIG;

Intelligent Image Capture:

typedef struct
{

DWORD	dwStructSize;	Size of HHP_INTEL_IMG structure in bytes.
DWORD	dwAspectRatio;	Ratio of barcode height to minimum barcode element width.
LONG	nOffsetX;	Offset from barcode center to the image window center in X direction, relative to barcode center in minimum barcode units.
LONG	nOffsetY;	Offset from barcode center to the image window center in Y direction, relative to barcode center in minimum barcode units.
DWORD	dwWidth;	Width of image in minimum barcode units.
DWORD	dwHeight;	Height of image in minimum barcode units.
SIZE	maxImgSize;	Maximum width and height for resulting intelligent image in pixels. Image size is guaranteed not to exceed either dimension.
BOOL	bSendBinary;	Have reader binarize data before transfer.

} IntelIImgDesc_t, HHP_INTEL_IMG, *PHHP_INTEL_IMG;

Serial Port Config:

typedef struct _tagHHP_SERIAL_PORT_CONFIG
{

DWORD	dwStructSize;	Size of HHP_SERIAL_PORT_CONFIG structure in bytes.
HHP_BAUD_RATE	baudRate;	Baud rate for connection. NOTE: Reader always powers up at 115200. The 5X00 Series changes the baud rate to this value once it connects at 115200. Also, the SDK will not allow you to connect at bauds above 115200 for WinCE unless the port driver is sio950.dll (high speed driver for sio950 chipset).
HHP_DATA_BITS	dataBits;	Number of data bits.
HHP_PARITY	parity;	Parity.
HHP_STOP_BITS	stopBits;	Number of stop bits.
BOOL	bAutoBaud;	Not supported in the IT4500.

} HHP_SERIAL_PORT_CONFIG, *PHHP_SERIAL_PORT_CONFIG;

Imager capabilities structure – Items are read only:

typedef struct _tagHHP_IMAGER_CAPS
{

DWORD	dwStructSize;	Size of HHP_IMAGE_CAPS structure in bytes.
SIZE	fullImgSize;	Size of image captured by imager before cropping and subsampling.
DWORD	dwImgrBpp;	Bits per pixel of image captured by imager.
DECODER_TYPE	decoderType;	Level of symbology support in the imager's decoder.
IMAGER_TYPE	imagerType;	Decoded Out Serial, Decoded Out USB, Non-Decoded Out.

} HHP_IMAGER_CAPS, *PHHP_IMAGER_CAPS;

Imager Type (decoded out or non decoded out)

typedef enum
{

HHP_UNKNOWN_IMAGER	Unable to determine engine type.
HHP_DECODED_IMAGER_4080	Serial (RS-232) 4080 imager with internal decoder.
HHP_DECODED_IMAGER_4080_USB	USB serial 4080 imager with internal decoder
HHP_NON_DECODED_IMAGER_4000	Incorporated 4000 imager.
HHP_DECODED_IMAGER_5080VGA	USB serial 5080 VGA imager with internal decoder.
HHP_DECODED_IMAGER_5080VGA_USB	Serial (RS-232) 5080 VGA imager with internal decoder and PSOC.
HHP_NON_DECODED_IMAGER_5000VGA	Incorporated 5000 VGA imager.

HHP_DECODED_IMAGER_5080VGA_PSOC	Serial (RS-232) 5080 imager with internal decoder.
HHP_DECODED_IMAGER_5080VGA_PSOC_USB	USB serial 5080 imager with internal decoder.
HHP_NON_DECODED_IMAGER_5000VGA_PSOC	Incorporated 5000 VGA imager with PSOC.
HHP_DECODED_IMAGER_5080	Serial (RS-232) 5080 imager with internal decoder.
HHP_DECODED_IMAGER_5080_USB	USB serial 5080 imager with internal decoder.
HHP_NON_DECODED_IMAGER_5000	Incorporated 5000 imager.
HHP_DECODED_IMAGER_5080_PSOC	Serial (RS-232) 5080 imager with internal decoder and PSOC.
HHP_DECODED_IMAGER_5080_PSOC_USB	USB serial 5080 imager with internal decoder and PSOC.
HHP_NON_DECODED_IMAGER_5000_PSOC	Incorporated 5000 imager with PSOC.

} ImagerType_t, IMAGER_TYPE;

5 OEM-Configurable SDK Functionality

The 5X80 accommodates a variety of PDA/PDT, static mount, and "gun" type installations, so the 5X00 Series has included the flexibility to modify some SDK behaviors. You can provide a DLL that affords the SDK access to the hardware trigger, hardware sleep lines, and access to the open driver handle and registry entries, which can be used to specify high speed baud rates (greater than 115200). You can also specify whether to force a Y-modem communications protocol for image transfers.

OEM Supplied DLL

The OEM DLL interface is described in SDK header file OemDll.h. The DLL can export some or all of the available functionality. If the DLL is named ImgrHwrLines.dll and is located in the default library search path, it will be loaded automatically by the SDK. Otherwise, it can by loaded manually by calling the SDK function hhpSetHardwareLineDllFileName. The DLL exports are described in the following list:

ConfigureCommPort

ConfigureCommPort performs any special driver configuration that needs to be done to support the specified communication port settings. Configuring the communications driver requires the current driver handle, so you must also support SetCommDriverHandle(). The function is called by the SDK whenever you set/change the communication port settings. It is called after the SDK is finished processing pConfig, so any changes you make will not be overridden.

bool ConfigureCommPort(
PCommPortConfig pConfig
)

Parameter	Description
pConfig	The communication settings requested. The CommPortConfig structure contains: Baud Rate, Data Bits, Parity, Stop Bits and RTS control.

ImagerPoweredDown

ImagerPoweredDown checks the imager powered down hardware line for the imager power state. The powered down hardware line is active high. ImagerPoweredDown returns true if the hardware line is high, false if it is low.

bool ImagerPoweredDown(
void
)

ModifyCommPortDCB

ModifyCommPortDCB receives the RS-232 configuration (DCB) structure just before it's sent to the serial driver. You can then modify the RTS/CTS and DTR/DTS settings if they are used for the imager's power or trigger.

void ModifyCommPortDCB(
LPDCB lpDCB
)

Parameter	Description
LPDCB lpDCB	Structure used by windows to configure a serial port.

SetCommDriverHandle

SetCommDriverHandle receives the handle to the open communications port. The handle can then be used to access private driver ioctl calls.

void SetCommDriverHandle(
HANDLE hDriver
)

Parameter	Description
hDriver	The current handle to the open driver used to communicate with the imager.

SetHardwareTrigger

SetHardwareTrigger triggers the imager hardware trigger line, depending on bEnable. The hardware trigger line is active low.

void SetHardwareTrigger(
bool bEnable
)

Parameter	Description
bEnable	If true, set trigger line low to trigger imager. If false, set trigger line high to turn off trigger.

WakeUpImager

WakeUpImager toggles the imager hardware wakeup line from low to high, delay, (see Integration Manual for timing), then from high back to low.

void ImagerPoweredDown(
void
)

Registry Entries

There are two registry entries used to modify the SDK default behavior. The values are both located in the registry key HKEY_LOCAL_MACHINE\SOFTWARE\Hand Held Products\MatrixDemos.

Baud Rate

The SDK will not normally allow high speed serial connections (connections greater than 115200 baud) unless it recognizes the driver name in the registry. The first registry value, BaudRate, specifies the first baud rate at which the demos will attempt to connect. It also overrides the normal block that sets high speed baud rates. This lets you specify a baud rate greater than 115200.

ForceHmodem

The SDK does not normally use its Y-Modem variant, Hmodem, if a high speed serial connection is specified. However, if your driver does not support true hardware RTS flow control, there is a substantial risk of communication buffer overruns. These overruns can cause data loss, especially during image transfer. The registry value ForceHmodem lets you force the use of the communications protocol, preventing data buffer overflow.

6 Program Samples

Configuration Management

Sample 1 - Set code 39 defaults turning on Full ASCII

```
CODE39_Tcode39;                                    // Structure for Code 39.
TCHAR   tcErrMsg[ 128 ];                           // Error message buffer.
Result_tnResult = RESULT_ERR_INTIALIZE;            // Return code.

code39.dwStructSize = sizeof( CODE39_T );          // setup size parameter, used in
                                                     struct verification.
code39. dwMask = SYM_MASK_FLAGS;                   // you want all info.
if( (nResult = hhpReadSymbologyConfig( SETUP_TYPE_DEFAULT,SYM_CODE39,&code39 )) ==
RESULT_SUCCESS )
{
        code39.dwFlags |= SYMBOLOGY_ENABLE_FULLASCII;// OR flags with Enable Full ASCII
                                                       flag to turn on code 39.
        nResult = hhpWriteSymbologyConfig( SYM_CODE39,&code39 );
}
hhpGetErrorMessage(  nResult,tcErrMsg );
_tprintf( _T("Setup Code39 Returned: %s\n"),tcErrMsg;
```

Sample 2 - Set the capture mode to photo image

Note: This changes the imager configuration for the items selected. The imager uses these values if they are not overridden at the time of image capture/transfer.

```
HHP_IMAGE_TRANSFER  imgTrans;                      // Image transfer structure.
TCHAR               tcErrMsg[ 128 ];               // Error message buffer.
Result_t            nResult = RESULT_ERR_INTIALIZE; // Return code.

// Set the structure size and structure mask
imgTrans.dwStructSize = sizeof( HHP_IMAGE_TRANSFER);
imgTrans.dwMask = IMAGE_CAPTURE_MODE_MASK;

// Turn on photo image mode.
ImgTrans.captureMode = HHP_AUTOEXPOSURE_PHOTO;

// Call the write configuration function specifying the HHP_ACQUISITION_STRUCTURE.
nResult = hhpWriteConfigItem( IMAGE_ACQUISITION,&imgTrans );

// Display error code (NOTE: RESULT_SUCCESS error code returns string SUCCESS.
hhpGetErrorMessage(  nResult,tcErrMsg );
_tprintf( _T("Change Imager Config To Photo Capture Mode: %s\n"),tcErrMsg;
```

Barcode Capture

Sample 3 - A synchronous barcode capture

```
HHP_DECODE_MSG      decodeInfo;
TCHAR               tcErrMsg[ 128 ];                // Error message buffer.
Result_t            nResult = RESULT_ERR_INTIALIZE; // Return code.

 // Make sure to set the structure size!
 decodeInfo.dwStructSize = sizeof( HHP_DECODE_MSG );
 // Call the SDK function to capture a barcode setting the bWait parameter to TRUE, 6
 second timeout.
 if ( (nResult = hhpCaptureBarcode( &decodeInfo,6000,TRUE ) == RESULT_SUCCESS )
 {
       _tprintf( _T("Barcode: %s\n"),decodeInfo.pchMessage );
       _tprintf( _T("Barcode Length: %d\n"),decodeInfo.nLength );
       _tprintf( _T("AIM Id : %cn"),decodeInfo.chCodeID );
       _tprintf( _T("HHP Id: %cn"),decodeInfo.chSymLetter );
       _tprintf( _T("Symbol Modifier: %c\n"),decodeInfo.chSymModifier );
 }
 else
 {
       hhpGetErrorMessage(  nResult,tcErrMsg );
       _tprintf( _T("Capture Barcode Returned: %s\n"),tcErrMsg );
 }
```

Sample 4 - An asynchronous barcode capture using an event

```
HHP_DECODE_MSG      decodeInfo;                     // Returned decoded data message
                                                    structure.
hhpEventType_t      eventType = HHP_BARCODE_EVENT;  // Type of event that occurred.
TCHAR               tcErrMsg[ 128 ];                // Error message buffer.
Result_t            nResult = RESULT_ERR_INTIALIZE; // Return code.

// Verify the event is valid (or you won't get any notification)
If( hEvent != NULL )
{
 // Register the event with the SDK.
 if ( (nResult = hhpSetAsyncMethods( hEvent,NULL,NULL ) == RESULT_SUCCESS )
 {
       // Call the SDK function to capture a barcode setting the bWait parameter to FALSE,
            6 second timeout.
       if ( (nResult = hhpCaptureBarcode( NULL,6000,FALSE ) == RESULT_SUCCESS )
       {
           // Make sure to set the structure size!
           decodeInfo.dwStructSize = sizeof( HHP_DECODE_MSG );
           // Wait on event being set by SDK then call SDK to get results.
           if( WaitForSingleObject( hEvent,7000 ) == WAIT_OBJECT_0 )
           nResult = hhpGetAsyncResult( &event,&decodeInfo );
       }
 }
}
if( nResult == RESULT_SUCCESS )
{
    _tprintf( _T("Barcode: %s\n"),decodeInfo.pchMessage );
    _tprintf( _T("Barcode Length: %d\n"),decodeInfo.nLength );
    _tprintf( _T("AIM Id : %cn"),decodeInfo.chCodeID );
    _tprintf( _T("HHP Id: %cn"),decodeInfo.chSymLetter );
    _tprintf( _T("Symbol Modifier: %c\n"),decodeInfo.chSymModifier );
```

```
}
else
{
    hhpGetErrorMessage(  nResult,tcErrMsg );
    _tprintf( _T("Capture Barcode Returned: %s\n"),tcErrMsg );
}
```

Sample 5 - An asynchronous barcode capture message notification

Note: You must hook the message WM_HHP_EVENT_HWND_MSG in your message loop to receive a barcode event notification. In this example, the message is hooked to call OnEventMsg.

```
TCHAR        tcErrMsg[ 128 ];                  // Error message buffer.
Result_t     nResult = RESULT_ERR_INTIALIZE;   // Return code.
HWND         hWnd = GetSafeHwnd();             // The window to which message is to be
                                                   sent. (note: this example is MFC c++)

// Register the message window with the SDK.
if ( (nResult = hhpSetAsyncMethods( NULL,hWnd,NULL ) == RESULT_SUCCESS )
{
 // Call the SDK function to capture a barcode, 6 second timeout.  Unless call fails, you
 will get a message when command completes.
 if ( (nResult = hhpCaptureBarcode( NULL,6000,FALSE ) != RESULT_SUCCESS )
 {
       hhpGetErrorMessage(  nResult,tcErrMsg );
       _tprintf( _T("Capture Barcode Returned: %s\n"),tcErrMsg );
 }
}

// Message Handler function
LRESULT OnEventMsg( WPARAM wParam,LPARAM lParam )
{
 hhpEventTye_t         eventType = (hhpEventType_t)wParam; // Event type
 DWORD                 dwBytes = lParam;                   // Number of bytes in barcode.
                                                           You don't actually use it
                                                           here.
 HHP_DECODE_MSG        decodeInfo;                         // Decode message structure.
 TCHAR                 tcErrMsg[ 128 ];                    // Error message buffer.
 Result_t              nResult = RESULT_ERR_INTIALIZE;     // Return code.

 // Verify the event type is barcode
 if( eventType == HHP_BARCODE_EVENT )
 {
       // Make sure to set the structure size!
       decodeInfo.dwStructSize = sizeof( HHP_DECODE_MSG );
       if( (nResult = hhpGetAsyncResult( &eventType,&decodeInfo )) == RESULT_SUCCESS )
       {
           _tprintf( _T("Barcode: %s\n"),decodeInfo.pchMessage );
           _tprintf( _T("Barcode Length: %d\n"),decodeInfo.nLength );
           _tprintf( _T("AIM Id : %cn"),decodeInfo.chCodeID );
           _tprintf( _T("HHP Id: %cn"),decodeInfo.chSymLetter );
           _tprintf( _T("Symbol Modifier: %c\n"),decodeInfo.chSymModifier );
       }
       else
       {
           hhpGetErrorMessage(  nResult,tcErrMsg );
           _tprintf( _T("Capture Barcode Returned: %s\n"),tcErrMsg );
       }
 }
}
```

Sample 6 - An asynchronous barcode capture using Callback function

```
// Message Handler function
BOOL CALLBACK EventCallback(HHP_EVENT_TYPE eventType,DWORD dwBytes )
{
 HHP_DECODE_MSG   decodeInfo;                      // Decode message structure.
 TCHAR            tcErrMsg[ 128 ];                 // Error message buffer.
 Result_t         nResult = RESULT_ERR_INTIALIZE;  // Return code.

 // Verify the event type is barcode
 if( eventType == HHP_BARCODE_EVENT )
 {
       // Make sure to set the structure size!
       decodeInfo.dwStructSize = sizeof( HHP_DECODE_MSG );
       // Retrieve the barcode event data.
       If( (nResult = hhpGetAsyncResult( &eventType,&decodeInfo )) == RESULT_SUCCESS )
       {
           _tprintf( _T("Barcode: %s\n"),decodeInfo.pchMessage );
           _tprintf( _T("Barcode Length: %d\n"),decodeInfo.nLength );
           _tprintf( _T("AIM Id : %cn"),decodeInfo.chCodeID );
           _tprintf( _T("HHP Id: %cn"),decodeInfo.chSymLetter );
           _tprintf( _T("Symbol Modifier: %c\n"),decodeInfo.chSymModifier );
       }
       else
       {
           hhpGetErrorMessage(  nResult,tcErrMsg );
           _tprintf( _T("Capture Barcode Returned: %s\n"),tcErrMsg );
       }
 }
}
// Code snippet to capture barcode
TCHARtcErrMsg[ 128 ];                             // Error message buffer.
Result_tnResult = RESULT_ERR_INTIALIZE;           // Return code.

// Register the callback function with the SDK.
if ( (nResult = hhpSetAsyncMethods( NULL,NULL,&EventCallback ) == RESULT_SUCCESS )
{
 // Call the SDK function to capture a barcode, 6 second timeout.  Unless call fails, we
 will get a message when command completes.
 if ( (nResult = hhpCaptureBarcode( NULL,6000,FALSE ) != RESULT_SUCCESS )
 {
       hhpGetErrorMessage(  nResult,tcErrMsg );
       _tprintf( _T("Capture Barcode Returned: %s\n"),tcErrMsg );
 }
}
```

Image Capture

Sample 7 - A synchronous image capture

```
// Capture image specifies:
for capture - Photo Mode and Lights On During Frames.
for Transfer - Subsample value of 1 and Lossless transfer.
Note: You don't really have to pass anything except the HHP_IMAGE structure as long as you
want the imager config settings for capture and transfer.  Also, no progress feedback will
be received.

HHP_IMAGE          image;                          // Structure to hold captured image
HHP_IMAGE_TRANSFER imgTrans;                       // Image transfer options (override
                                                   imager config)
                                                   // Image capture options (override
                                                   imager config)
TCHAR              tcErrMsg[ 128 ];                // Error message buffer
Result_t           nResult = RESULT_ERR_INTIALIZE; // Return code

// Make sure to set imgAcqu structure size!
imgAcqu.dwStructSize = sizeof( HHP_IMAGE_ACQUISITION );

// Set the mask to activate captureMode and illuminatCycle (lights).
ImgAcqu.dwMask = (ILLUMINATION_DUTY_CYCLE_MASK | IMAGE_CAPTURE_MODE_MASK);

// Set values
ImgAcqu.captureMode = HHP_AUTOEXPOSURE_PHOTO;
ImgAcqu. IlluminatCycle = HHP_DUTY_CYCLE_ON;

// Make sure to set imgTrans structure size!
imgTrans.dwStructSize = sizeof( HHP_IMAGE_TRANSFER );

// Set the subsample and compression masks.
ImgTrans.dwMask = (SUBSAMPLE_VALUE_MASK | COMPRESSION_MODE_MASK);

// Set values
imgTrans.dwSubSample = 1;
imgTrans. compressionMode = COMPRESSION_LOSSLESS;

// Set the HHP_IMAGE structure size and allocate a buffer for the data, set the buffer
size and how you want to receive the data in the buffer.
Image.dwStructSize = sizeof( HHP_IMAGE );
Image.puchBuffer = new BYTE[ 324000 ];          // Allocate a buffer big enough to hold
                                                640x480x8 plus header (if BMP)
Image.nBufferSize = 324000;                     // SDK wants to know how big the buffer
                                                so no overflow.
Image.fileFormat = FF_RAW_GRAY;                 // 8 bit raw data.

// Call the SDK function to capture an image setting the bWait parameter to TRUE for
blocking/synchronous behavior.
if ( (nResult = hhpAcquireImage( &image,&imgTrans, &imgAcqu,TRUE ) == RESULT_SUCCESS )
{
      // Display the image.  NOTE: you could specify FF_BMP_GRAY.  The data would come as
      a BMP file.
}
else
{
     hhpGetErrorMessage(  nResult,tcErrMsg );
     _tprintf( _T("Image Capture Failed: %s\n"),tcErrMsg );
```

```
}
// Remember to delete your buffer.
delete [ ] image.puchBuffer;
```

Sample 8 - An asynchronous image capture using Windows Messaging

Note: You must hook the message WM_HHP_PROGRESS_HWND_MSG in your message loop to receive a barcode event notification. Here, assume the message is hooked to call OnEventMsg.

```
// Capture image specifies:
for capture - none (use imager config settings)
for transfer - subsample value of 2 and JPEG transfer, progress notification

HHP_IMAGE_TRANSFER  imgTrans;                       // Image transfer options (override
                                                    imager config)
TCHAR               tcErrMsg[ 128 ];                // Error message buffer.
Result_t            nResult = RESULT_ERR_INTIALIZE; // Return code.
extern   DWORD      g_dwPercentComplete = NULL;     // Percent complete updated by SDK.

// Make sure to set imgTrans structure size!
imgTrans.dwStructSize = sizeof( HHP_IMAGE_TRANSFER );

// Set the subsample, compression, and, since you're specifying lossy compression (JPEG),
add the compression factor masks. You are also asking to receive both a Windows message
with progress information, as well as having the SDK update a DWORD with the percentage
of transfer completion.  For the latter, it is important the DWORD not lose scope or be
deleted before the asynchronous call completes.  You might have a thread or timer that
looks at this value periodically before altering the user.

ImgTrans.dwMask =( SUBSAMPLE_VALUE_MASK | COMPRESSION_MODE_MASK | COMPRESSION_FACTOR_MASK
                    | TRANSFER_UPDATE_HWND | TRANSFER_UPDATE_DWORD);
// Set values
imgTrans.dwSubSample = 2;                           // Every other pixel and every other
                                                    row.
imgTrans.compressionMode = COMPRESSION_LOSSY;       // Image compression lossy which is
                                                    mode 1 JPEG lossy.
imgTrans.compressionFactor = 95;                    // Image compression factor (image
                                                    quality) of 95%.
ImgTrans. hTransferNotifyHwnd = GetSafeHwnd();      // Set this to the window handle you
                                                    wish to receive the progress message.
imgTrans.dwPercentComplete = &g_dwPercentComplete;  //Tell SDK to update this DWORD as
                                                    data is received.

// Call the SDK function to capture an image setting the bWait parameter to FALSE for
asynchronous behavior.
if ( (nResult = hhpAcquireImage( NULL,&imgTrans,NULL,FALSE ) == RESULT_SUCCESS )
{
     // Display the image.  NOTE: You could specify FF_BMP_GRAY.  The data would come as
     a BMP file.
}
else
{
    hhpGetErrorMessage(  nResult,tcErrMsg );
    _tprintf( _T("Image Capture Failed: %s\n"),tcErrMsg );
}

// Message Handler for transfer progress message
LRESULT OnProgressMsg( WPARAM wParam,LPARAM lParam )
{
     DWORDdwBytesSoFar = wParam;// Number of bytes receive to this point.
     DWORDdwBytesToRead = lParam;// Total number of bytes expected.
```

```
        return( 0 );
    }

    // Message Handler for image acquisition ended function (can be result of a failure as
    well)
    LRESULT OnEventMsg( WPARAM wParam,LPARAM lParam )
    {
     hhpEventTye_t  eventType = (hhpEventType_t)wParam;// Event type
     DWORD          dwBytes = lParam;                  // Number of bytes in barcode. You
                                                       don't actually use it here.
     HHP_IMAGE      Image;                             // Returned image structure.
     TCHAR          tcErrMsg[ 128 ];                   // Error message buffer
     Result_t       nResult = RESULT_ERR_INTIALIZE;    // Return code

     // Verify the event type is barcode
     if( eventType == HHP_IMAGE_EVENT )
     {
           // Set the HHP_IMAGE structure size and allocate a buffer for the data, set the
           buffer size and how we want to receive the data in the buffer.
           Image.dwStructSize = sizeof( HHP_IMAGE );
           Image.puchBuffer = new BYTE[ 324000 ];      // Allocate a buffer big enough to
                                                       hold 640x480x8 plus header (if BMP)
           Image.nBufferSize = 324000;                 // SDK wants to know how big the
                                                       buffer is so there's no overflow
           Image.fileFormat = FF_BMP_GRAY;             // 8 bit bmp file format data

           if( (nResult = hhpGetAsyncResult( &eventType,&Image )) == RESULT_SUCCESS )
           {
               // save image data to a bmp file and/or display it
           }
           else
           {
               hhpGetErrorMessage(  nResult,tcErrMsg );
               _tprintf( _T("Capture Image Failed: %s\n"),tcErrMsg );
           }
     }
     // Remember to delete your buffer
     delete [ ] image.puchBuffer;

     return( 0 );
}
```

7 Customer Support

Product Service and Repair

Honeywell International Inc. provides service for all its products through service centers throughout the world. To obtain warranty or non-warranty service, contact the appropriate location below to obtain a Return Material Authorization number (RMA #) before returning the product.

North America

Telephone: (800) 782-4263
E-mail: hsmnaservice@honeywell.com

Latin America

Telephone: (803) 835-8000
Telephone: (800) 782-4263
Fax: (239) 263-9689
E-mail: laservice@honeywell.com

Brazil

Telephone: +55 (11) 5185-8222
Fax: +55 (11) 5185-8225
E-mail: brservice@honeywell.com

Mexico

Telephone: 01-800-HONEYWELL (01-800-466-3993)
Fax: +52 (55) 5531-3672
E-mail: mxservice@honeywell.com

Europe, Middle East, and Africa

Telephone: +31 (0) 40 2901 633
Fax: +31 (0) 40 2901 631
E-mail: euroservice@honeywell.com

Hong Kong

Telephone: +852-29536436
Fax: +852-2511-3557
E-mail: apservice@honeywell.com

Singapore

Telephone: +65-6842-7155
Fax: +65-6842-7166
E-mail: apservice@honeywell.com

China

Telephone: +86 800 828 2803
Fax: +86-512-6762-2560
E-mail: apservice@honeywell.com

Japan

Telephone: +81-3-6730-7344
Fax: +81-3-6730-7222
E-mail: apservice@honeywell.com

Online Product Service and Repair Assistance

You can also access product service and repair assistance online at www.honeywellaidc.com.

Technical Assistance

If you need assistance installing or troubleshooting your device, please call your distributor or the nearest technical support office:

North America/Canada

Telephone: (800) 782-4263
E-mail: hsmnasupport@honeywell.com

Latin America

Telephone: (803) 835-8000
Telephone: (800) 782-4263
E-mail: hsmlasupport@honeywell.com

Brazil

Telephone: +55 (11) 5185-8222
Fax: +55 (11) 5185-8225
E-mail: brsuporte@honeywell.com

Mexico

Telephone: 01-800-HONEYWELL (01-800-466-3993)
E-mail: soporte.hsm@honeywell.com

Europe, Middle East, and Africa

Telephone: +31 (0) 40 7999 393
Fax: +31 (0) 40 2425 672
E-mail: hsmeurosupport@honeywell.com

Hong Kong

Telephone: +852-29536436
Fax: +852-2511-3557
E-mail: aptechsupport@honeywell.com

Singapore

Telephone: +65-6842-7155
Fax: +65-6842-7166
E-mail: aptechsupport@honeywell.com

China

Telephone: +86 800 828 2803
Fax: +86-512-6762-2560
E-mail: aptechsupport@honeywell.com

Japan

Telephone: +81-3-6730-7344
Fax: +81-3-6730-7222
E-mail: aptechsupport@honeywell.com

Online Technical Assistance

You can also access technical assistance online at www.honeywellaidc.com.

Honeywell Scanning & Mobility
9680 Old Bailes Road
Fort Mill, SC 29707

www.honeywellaidc.com

5X10-80SDK-UG Rev D
4/11