5 Printer Support

The 700 Series Color Mobile Computer works with the following printers from Intermec Technologies. Contact an Intermec Representative for information about these printers.

- **6820**
  A full-page, 80-column printer.
- **6808**
  A 4-inch belt-mount printer.
- **781T**
  A 2-inch belt-mount printer with a Bluetooth compatible module from Socket Communications.
- **782T**
  A 2-inch workboard printer.

Printing ASCII

The following methods for printing using Pocket PC at this time is as follows:

- Add port drivers to print ASCII directly to the port.
- Use LinePrinter ActiveX Control from the Software Developer's Kit (SDK) - *see the SDK User's Manual for more information.*
- Via wireless printing - *see the Wireless Printing Development Guide on the 700C Software Tools CD for more information.*

Directly to a Port

Printing directly to the port sends RAW data to the printer. The format of this data depends upon your application and the printer capabilities.

You must understand the printer commands available for your specific printer. Generally, applications just send raw ASCII text to the printer. Since you are sending data to the printer from your application directly to the port you are in complete control of the printers operations. This allows you to do line printing *(print one line at a time)* rather than the page format printing offered by the GDI approach. It is also much faster since data does not have to be converted from one graphics format to the other (display to printer). Most Intermec® printers use Epson Escape Sequences to control print format operations.

These commands are available in documentation you receive with your printers or from technical support. Win32 APIs are required to print directly to the port.

Directly to a Generic Serial Port

To print directly to a generic serial port printer (non-Intermec printers):

- Use CreateFile() to open ports - COM1: can be opened on most devices.
- Use WriteFile() to send data directly to the printer.
- Use CloseHandle() when you are finished printing to close the port.

IrDA Printer Driver

IrDA printing is only available on the certain devices and is supported directly by the Windows CE load via the IrSock API provided by the Microsoft Win32 API without need for additional drivers. Intermec 6804, 6805, 6806, 6808 and 6820 and other IrDA printers are supported.

NPCP Printer Driver

The NPCP printer communications driver (NPCPPORT.DLL) is a Stream Device Driver built into the operating system. The driver supports only NPCP communications to and from the 6820 and 4820 printers over a selected serial port.

All applications use WIN32 API functions to access the drivers. Basic operations are easily implemented by applications through the CreateFile(), WriteFile(), ReadFile(), DeviceIOControl(), and CloseHandle() Win32 APIs.

Operations to upgrade printer modules, perform printer diagnostics, and get printer configuration are performed largely via DeviceIOControl() functions.

About NPCP

NPCP (Norand® Portable Communications Protocol) is a proprietary protocol that provides session, network, and datalink services for Intermec mobile computers in the Intermec LAN environment used with printers and data communications.

NPCP Driver Installation and Removal

Use LPT9: for the NPCP printer device and COM1 for the last parameter. COM1 is the connection available via the 700 Series Computer.

Applications use the RegisterDevice() function to install the driver. DeregisterDevice() uninstalls the device driver and frees memory space when the driver is not required. Use the HANDLE returned by RegisterDevice() as the parameter to DeregisterDevice().

Use the RegisterDevice() function call as demonstrated below. Specify the full path name to the driver starting at the root for the RegisterDevice() function to work properly. The last parameter to RegisterDevice() is a DWORD that represents the name of the port for the NPCP stream driver to use. Build this parameter on the stack if it is not to be paged out during the call. The first parameter "LPT" (Device Name) and the second parameter "9' (index), indicate the name of the registered device, such as LPT9. This is used in the CreateFile() function call.

```
Install()
{
  HANDLE hDevice;
  TCHAR port[6];
  port[0] = TCHAR('C');
  port[1] = TCHAR('O');
  port[2] = TCHAR('M');
  port[3] = TCHAR('1');
  port[4] = TCHAR(':');
  port[5] = TCHAR(0);
  hDevice = RegisterDevice ( (TEXT("LPT"), 9,
  TEXT("\\STORAGE CARD\\WINDOWS\\NPCPPORT.dll"), (DWORD)port);
}
```

Opening the NPCP Driver

The application opens the NPCP driver by using the CreateFile() function. The call can be implemented as follows. The first parameter "LPT9:" must reflect the device name and index used in the RegisterDevice() function call and will fail for any of the following reasons:

```
hFile = CreateFile(_T("LPT9:"), GENERIC_WRITE |
GENERIC_READ, 0, NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL,
NULL);
```

- The port associated with the device during RegisterDevice() is in use.
- The NPCP device is already open.
- The share mode is not set to zero. The device cannot be shared.
- Access permissions are not set to GENERIC_WRITE | GENERIC_READ. Both modes must be specified.

Closing the NPCP Driver

Using the CloseHandle() (hFile) function closes the NPCP driver. Where *hFile* is the handle returned by the CreateFile() function call.

- TRUE = the device is successfully closed.
- FALSE = an attempt to close NULL HANDLE or an already closed device.

Reading from the NPCP Driver

Reading of the NPCP printers is not supported since all responses from the printer are the result of commands sent to the printer. DeviceIoControl() functions are provided where data is to be received from the printer.

Writing to the NPCP Driver

All Print data can be sent to the printer using the WriteFile() function. The print data written to the driver must contain the proper printer commands for formatting. If the function returns FALSE, the NPCP error may be retrieved using IOCTL_NPCP_ERROR. See the description on the next page.

NPCP Driver I/O Controls

An application uses the DeviceIoControl() function to specify an printer operation to be performed. Certain I/O controls are required to bind and close communication sessions with the printer, and must be completed before any other commands to the driver can execute properly.

The function returns TRUE to indicate the device successfully completed its specified I/O control operation, otherwise it returns FALSE. The following I/O control codes are defined:

```
#define IOCTL_NPCP_CANCEL
CTL_CODE(FILE_DEVICE_SERIAL_PORT,0x400,METHOD_BUFFERED,FILE_ANY_ACCESS)
#define IOCTL_NPCP_BIND
CTL_CODE(FILE_DEVICE_SERIAL_PORT,0x401,METHOD_BUFFERED,FILE_ANY_ACCESS)
#define IOCTL_NPCP_CLOSE
CTL_CODE(FILE_DEVICE_SERIAL_PORT,0x402,METHOD_BUFFERED,FILE_ANY_ACCESS)
#define IOCTL_NPCP_ERROR
CTL_CODE(FILE_DEVICE_SERIAL_PORT,0x403,METHOD_BUFFERED,FILE_ANY_ACCESS)
#define IOCTL_NPCP_FLUSH
CTL_CODE(FILE_DEVICE_SERIAL_PORT,0x404,METHOD_BUFFERED,FILE_ANY_ACCESS)
#define IOCTL_NPCP_IOCTL
CTL_CODE(FILE_DEVICE_SERIAL_PORT,0x405,METHOD_BUFFERED,FILE_ANY_ACCESS)
#define IOCTL_NPCP_PRTVER
CTL_CODE(FILE_DEVICE_SERIAL_PORT,0x406,METHOD_BUFFERED,FILE_ANY_ACCESS)
```

- **IOCTL_NPCP_CANCEL**
  This cancels all printing at the printer. It flushes the printer buffers and reinitializes the printer to its default state. No parameters are required.
- **IOCTL_NPCP_BIND**
  This command is required before any data is sent or received by the printer. Once the driver is opened, the application must bind the communications session with the printer before any data can be sent or received by the printer. If an error occurs during the bind, the application may use IOCTL_NPCP_ERROR to get the current extended error code. No parameters are required.
- **IOCTL_NPCP_CLOSE**
  This command closes the current session with the printer. This function always returns TRUE. No parameters are required.
- **IOCTL_NPCP_ERROR**
  This command returns the extended NPCP error code in PL/N format. The word returned will contain the PL/N compatible error code in the low byte and completion flags in the high byte. If the frame that returned an error was not received correctly by the printer the FRAME_NOT_ACKED bit will be set in the high byte. This operation always returns TRUE. An output buffer of at least 2 bytes is required. See "*NPCP Error Codes*" on page 133.
- **IOCTL_NPCP_FLUSH**
  This command allows the application to poll the printer for errors while the report is completing the print process at the printer. If an error occurs during the polling process, the operation will return FALSE and the application can get the extended error code by using IOCTL_NPCP_ERROR. No parameters are required.

NPCP Printer Communications

All NPCP printer communications should be based on the following flow:

1. Use CreateFile(); to open the printer driver.
2. Use IOCTL_NPCP_BIND to bind a session with the printer; IOCTL_NPCP_ERROR to check for errors on the bind to ensure success; and IOCTL_NPCP_CANCEL to cancel any outstanding print jobs.
3. Use IOCTL_NPCP_FLUSH to poll the printer to free up printer buffer resources. Use IOCTL_NPCP_FLUSH to poll the printer's status. If an error is reported by the IOCTL, then use IOCTL_NPCP_ERROR to get the error and determine the correct recovery procedure.
4. Use WriteFile(); to write your data to the printer. Check for errors and that all data were written. Use IOCTL_NPCP_ERROR to get the extended error. If the error is critical in nature, use IOCTL_NPCP_CLOSE, followed by CloseFile(), to end the communications session. Start a new session, beginning with step 1 to ensure proper printing. For noncritical errors display the error and retry the operation.
5. After all data is sent to the printer, ensure that the printer continues to print the report properly by polling the printer's status. Use IOCTL_NPCP_FLUSH to poll the printer's status. If an error is reported by the IOCTL, then use IOCTL_NPCP_ERROR to get the error and determine the correct recovery procedure.

Sample Code

See sample code in the "\700 Color Dev Tools\Installable Drivers\Port Drivers\Npcp\NPCPPrint\" directory for more details on printing, printer communications and error code handling.

NPCP Error Codes

Call the IOCTL_NPCP_ERROR I/O control function to receive PL/N compatible error codes. Applications must decide how to act upon the data returned.

```
// Definition of NPCP communications Errors and Printer Errors
#define PNRDY (BYTE)102 // link not ready error
#define RXTMO (BYTE)104 // link no receive error
#define TXTMO (BYTE)106 // link no transmit error
#define BADADR (BYTE)111 // frame address error
#define GAPERR (BYTE)112 // link gap error (timeout) in receive data
#define LSRPE (BYTE)113 // frame parity error on length field
#define IFTS (BYTE)120 // session layer - invalid frame this state
#define NS_NE_VR (BYTE)121 // session layer sequence error
#define NR_NE_VS (BYTE)122 // session layer sequence error
#define MAC_CRCERR (BYTE)124 // MAC CRC error
#define RLENERR (BYTE)123 // MAC too much data received
#define FRMERR (BYTE)200 // Frame Reject
#define FRMERR_IF (BYTE)201 // Frame Reject - Invalid Frame
#define FRMERR_NR (BYTE)202 // Frame Reject - NR Mismatch
#define FRMERR_NS (BYTE)203 // Frame Reject - NS Mismatch
#define NDMERR (BYTE)204 // Normal Disconnect mode error
#define BINDERR (BYTE)210 // bind error
#define IPLDUR (BYTE)221 // invalid presentation layer response
#define HEADJAM (BYTE)222 // printer head jam
#define PAPEROUT (BYTE)223 // printer paper out
#define LOWVOLTS (BYTE)224 // printer low voltage
#define HIVOLTS (BYTE)225 // printer over voltage
#define LOWBAT (BYTE)226 // printer low battery
#define COVEROFF (BYTE)227 // printer cover off error
#define HEADFAULT (BYTE)228 // printer head short or driver short error
#define PFFAULT (BYTE)229 // paper feed motor fault.
#define FRAME_NOT_ACKED 0x8000 // frame was not received by printer and need to
be resent.
```

O' Neil Printer Driver

The DTR printer communications driver is a Stream Device Driver named ONEIL.DLL.

All applications use WIN32 API functions to access drivers. Basic operations are easily implemented by applications through the CreateFile(), WriteFile(), DeviceIOControl() and CloseHandle() Win32 APIs.

The driver supports communications to 6804DM, 6804T, 6805A, 6806, 6808, 681T, and 781 printers over a selected serial port.

DTR Driver Installation and Removal

Your application must install the device driver by using the RegisterDevice() function. The driver name is ONEIL.DLL. We recommend that you use "DTR" for the Device Name parameter, "1" for the Device Driver index parameter, and use any of the following strings for the last parameter:

- NULL (==0) Defaults to COM1 @ 9600
- "COM1" only COM port specified defaults to 9600
- "COM1:9600" sets to COM port and specified bit rate
- "COM1:19200" sets to COM port and specified bit rate

Use the HANDLE returned by RegisterDevice() as the parameter to DeregisterDevice(). The correct usage of the RegisterDevice() function call is demonstrated below. You may use DeregisterDevice() to uninstall the driver.

```
Install()
{

   HANDLE hDevice;
   TCHAR port[6];
   port[0] = TCHAR('C');
   port[1] = TCHAR('O');
   port[2] = TCHAR('M');
   port[3] = TCHAR('1');
   port[4] = TCHAR(':');
   port[5] = TCHAR(0);
   hDevice = RegisterDevice ( (TEXT("DTR"), 1, TEXT("\\WINDOWS\\ONEIL.DLL"),
   (DWORD)port);

}
```

Opening the DTR Driver

The application opens the DTR driver by using the CreateFile() function. The call can be implemented as follows:

```
hFile = CreateFile(_T("DTR1:"), GENERIC_WRITE, 0, NULL,
OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
```

The first parameter "DTR1:" must reflect the device name and index used in the RegisterDevice() function call.

The function call will fail for any of the following reasons:

- The port associated with the device during RegisterDevice() is currently in use.
- The DTR device is already open.
- The share mode is not set to zero. The device cannot be shared.
- Access permissions are not set to GENERIC_WRITE.

Closing the DTR Driver

Using the CloseHandle() (hFile) function closes the DTR driver. Where *hFile* is the handle returned by the CreateFile() function call.

- TRUE indicates the device is successfully closed.
- FALSE indicates an attempt to close a NULL HANDLE or an already closed device.

Writing to the DTR Driver

You can use the WriteFile() function to send all Print data to the printer. The print data being written must contain the proper formatting printer commands.

DTR Printer Communications

All DTR printer communications should be based on the following flow:

1. Use CreateFile(); to open the printer driver.
2. Use WriteFile() to write your data to the printer. Check for errors and that all data were written.
3. Use CloseHandle() to close the driver.

6 Scanner Support

The 700 Series Color Mobile Computer is available with imaging or laser scanning technologies, including the following:

- **APS linear imager**:
  Reads 1D symbologies and PDF 417 bar codes. Linear imaging using Vista Scanning technology reads low-contrast bar codes, laminated bar codes, and bar codes displayed on CRT or TRT displays. This imaging uses harmless LEDs for illumination and does not require any warning labels. Vista Scanning is more reliable than lasers as it is a completely solid state with no moving parts or oscillating mirrors.
- **2D Imager**:
  This decodes several stacked 1D and 2D symbologies, including PDF 417 and Data Matrix without "painting." It can also read 1D codes from any orientation, for example the scan beam does not need to be aligned perpendicular to the symbol in order to read it. Photography is a secondary application; the lens in the device will favor bar code reading. Photos are 640x480, 256 gray-scale.
- **1D laser scanner**:
  Traditional laser scanner that decodes 1D bar codes.
- **PDF 417 laser scanner**:
  Higher speed laser scanner that can read PDF 417 labels by "painting" the label.

Scanner Control and Data Transfer

Note: To use the methods described below, enable Data Collection functionality on the 700 Computer using the bootloader configuration menu. See Chapter 3, "I*nstalling Applications"* for more information.

The Data Server and associated software provide several ways to manipulate scanner control and data transfer between the scanner subsystem and user applications:

- **Automatic Data Collection COM Interfaces**:
  These COM interfaces allow user applications to receive bar code data, and configure and control the bar code reader engine.
- **ITCAxBarCodeReaderControl functions**:
  These ActiveX controls allow user applications to collect bar code data from the scanner, to configure the scanner, and to configure audio and visual notification when data arrives. For more information, see the *SDK User's Manual*.
- **ITCAxReaderCommand functions**:
  Use these ActiveX controls to modify and retrieve configuration information using the reader interface commands. For more information, see the *SDK User's Manual*.
- **Scanning EasySet bar code labels**:
  You can use the EasySet bar code creation software from Intermec Technologies Corporation to print configuration labels. Scan the labels to change the scanner configuration and data transfer settings.

Automatic Data Collection COM Interfaces

Data collection configuration and functionality cannot be accessed by any means *(including control panel applets or remote management applications)* until after the 700 Series Computer has completed initialization, which occurs during a warm- or cold-boot or after a firmware upgrade.

When initialization is complete, the green LED on the 700 Series Computer stops flashing. Changes made to configuration settings remain after a warm boot. After a cold-boot, all configuration settings are reset to their defaults with the exception of scanner configurations, which remain except for the Symbology Identifier transmission option or the Preamble and Postamble strings. To reset all configuration settings to the factory defaults, the S9C scanner firmware must be reloaded.

The Automatic Data Collection (ADC) functions are accessed through custom COM interfaces. These interfaces allow the application to receive bar code data and configure and control the bar code reader engine. The COM interfaces include the following functions:

- IADC *(starting on page 151)*
- IBarCodeReaderControl *(starting on page 159)*
- IS9CConfig *(starting on page 172)*
- IS9CConfig2 *(starting on page 204)*
- IS9CConfig3 *(starting on page 216)*
- IImage Interface *(starting on page 221)*

Multiple ADC COM Object Support

A 700 Series Computer may have multiple reader engines to decode different types of ADC data. For example, a bar code reader engine decodes raw bar code data and passes it to a bar code reader COM object. An RFID reader engine decodes raw RFID tag data and passes it to an RFID tag reader COM object.

ADC COM interfaces are implemented as in-process COM objects. An instance of the ADC COM object creates a logical connection to access or control the reader engine. Specifically, the IBarCodeReadConfig or IBarCodeReaderControl COM objects can manage the bar code scanner configuration while the ADC COM object can gather data simultaneously. These ADC COM objects or connections can be created in a single application or multiple applications. Up to seven instances of a COM object can be created for a reader engine. For more information, see "*How to Create and Use the ADC COM Interfaces*" below.

For data collection features, ADC COM objects also provide for read ahead and non-read ahead data access and grid data editing.

How to Create and Use the ADC COM Interfaces

You can also use the Input Device Functions *(starting on page 149)* to create and use the ADC COM interfaces.

1. Create and initialize the in-process Bar Code Reader object using ITCDeviceOpen() *(see page 149)*. This function returns a COM Interface pointer to the Bar Code Reader Object created by the function.
2. Set the data grid if data filtering is desired *(default grid gives the application all the data)*. Below is a sample code of how to set the grid to accept Code 39 data that starts with the letter "A" and is not a reader command.

```
ITC_BARCODEREADER_GRID stBCGrid;
stBCGrid.stDIGrid.szDataMask = TEXT("A%s");
stBCGrid.stDDGrid.dwSymbologyMask = BARCODE_SYMBOLOGY_CODE39;
stBCGrid.dwDataSourceTypeMask = ITC_DATASOURCE_USERINPUT;
HRESULT hrStatus = pIBCControl->SetAttribute(
ITC_RDRATTR_GRID,
reinterpret_cast<BYTE *>(&stBCGrid),
sizeof(stBCGrid)
);
```

3. Issue a read to accept the bar code data. The timestamp, symbology, and data type are put into the ITC_BARCODE_DATA_DETAILS structure. Passing in a pointer to this structure is optional. The following sample code uses an infinite timeout.

```
ITC_BARCODE_DATA_DETAILS stBCDetails;
BYTE rgbBCData[1024]; // Buffer used to accept the bar code data
DWORD dwBytesReceived; // Number of bytes in the return data.
HRESULT hrStatus = pIBCControl->Read(
rgbBCData,
sizeof(rgbBCData),
&dwBytesReceived,
& stBCDetails,
INFINITE
);
```

4. Compile and link the application.

Read-Ahead Bar Code Data Access

The Bar Code Reader COM object delivers ADC data to the connection in read-ahead mode. In this mode, the data is queued until a COM connection is ready to read it. Read-ahead mode decouples reader device performance from the application performance. That is, data is read as fast as the user can scan it, independent of the connection processing load. No data will be scanned until the first Read() function is posted.

Grid Data Filtering

The virtual wedge retrieves scanned Automatic Data Collection (ADC) data and sends it to the keypad driver so that the 700 Series Computer can receive and interpret the data as keypad input. The data can be filtered so that only data conforming to a certain text pattern or symbology will be sent to an application. After the data is filtered, it can be edited by adding, deleting, or rearranging portions of the text or by extracting portions of text for further editing. To filter and edit data, you need to define the virtual wedge grid parameters.

- **Grid Processing**:
  Grid processing takes place in two steps:
  - **Compilation**:
    In which the user's grid expressions are checked for errors and reduced to a binary form for faster matching. This is done whenever the virtual wedge grid is set or changed by configuration software.
  - **Matching**:
    In which data is tested against the grids set in compilation. Matching can be performed multiple times after a compilation. The AIM symbology ID of the data being tested, including the enclosing angle brackets, must be prepended to the incoming data.

Syntax

The basic syntax of each grid expression is:

```
<symID> filter-expression= > editing-expression
```

where:

- **symID**
  Is the AIM symbology ID *(see the AIM Symbology ID Defaults table starting on page 219)*.
- **filterexpression**
  Is any character string that includes valid filter expression values *(see the "Filter Expression Values" table on the next page)*.
- **editing-expression**
  Is any character string that includes valid editing expression values *(see the "Editing Expression Values" table on page 144)*.

Filter Expression Values

A filter-expression can be any string of text containing the operators listed below.

Filter Expression Values

Operator	Meaning	Example
Any character string not containing the special characters: . ? [ ] { } or \ *(period, question mark, left/right brackets, left/right curly brackets, backslash).*	Match the string literally.	super20 matches super20
\c where c is any of the special characters: . ? [ ] { } or \ *(period, question mark, left/right brackets, left/right curly brackets, backslash)*	Remove any special meaning of c.	* matches *
. (period)	Any character.	. matches x
^ (tarot)	Anchor the match at the beginning of the string.	^abc matches abc, but not aabc
$ (dollar sign)	Anchor the match at the end of the string.	abc$ matches abc but not abcc
? (question mark)	Repeat the preceding expression zero or one time.	aa? matches a or aa
* (asterisk)	Repeat the preceding expression zero or more times.	ab*c matches ac, abc, abbc, etc.
+ (plus symbol)	Repeat the preceding expression one or more times.	ab+c matches abc, abbc, etc.
[characterclass]	A series of nonrepeating characters denoting a class.	[abcdefhijklmnopqrstuvwxyz] is the class of all lowercase alphas.
[rangel`rangeh]	A sequential range of nonrepeating characters denoting a class.	[a-z] is the class of all lowercase alphas.
[^characterclass]	Any character except those denoted by a character class.	[^a-z] matches a numeric digit or a punctuation mark.
[characterclass_tag]	[:alnum:] - Alphanumeric characters [:alpha:] - Alphabetic characters [:blank:] - Tab and space [:cntrl:] - Control characters [:digit:] - Numeric characters [:graph:] - All printable characters except space [:lower:] - Lowercase letters [:print:] - All printable characters [:punct:] - Punctuation [:space:] - White space characters [:upper:] - Uppercase letters [:xdigit:] - Hexadecimal digits	[[:alpha:]]* matches Dynaction, Selmer, or NewWonder but not Super20
{num}	Matches exactly *num* repetitions.	a{3} matches only aaa
{min,}	Matches at least *num* repetitions.	a{3,} matches aaaa but not aa

Filter Expression Values (continued)

Operator	Meaning	Example
{min,max}	A repetition operator like + or *, except the number of repetitions is specified by *min* and *max*.	[a-z]{1,3} matches a, ab, or aab, but not aabc
(expr1)|(expr2)	Matches *expr1* or *expr2*.	a|b matches a or b
(subexpression)	Grouping operator to consolidate terms into a subexpression, which can override the order of evaluation. The subexpression is available for later matching or editing by means of *index*, where *index* is between 1-9 and refers to the index-th group in the string, counting from left to right. *0* refers to the whole expression.	Overriding evaluation order: (ab)*c matches c, abc, ababc, etc. Back-referencing: (aa)bb\1 matches aabbaa.

Editing Expression Values

This table lists the valid operators for editing expressions.

Operator	Meaning	Example
\index	The *index-th* subexpression (reading left-right) in the matched string. *index* must be between 0`9. *\0* is the matched expression itself.	M([0-9]{6})= > \1 produces 270494 when M270494 is scanned, stripping off the first character.
& or \0	The matched expression itself.	M[0-9]{6}= > \0-Conn and M[0-9]{6}= > &-Conn both produce M270494-Conn when M270494 is scanned.
\xhh	A concise representation of the lower 256 characters in the Unicode set. When converted, this is still a 16-bit value.	\x0d inserts a carriage return.
any character string	Inserts any character string in the output string.	See previous examples.

- *<symID>* is optional. If present, only data in the indicated symbology is accepted.
- If the entire expression is blank, all data is passed unchanged. If = > *editing-expression* is omitted, then all data that passes through the filter is returned unchanged. If = > *editing expression* is present, the data is transformed by editing-expression.
- Multiple grid expressions can be compiled and are related in a logical OR fashion. These are expressed as single grid expressions separated by semicolons. When matching is attempted, the first grid expression from left to right that achieves a match will cause the data to be accepted.
- All pattern expressions and parsed data are in Unicode.

Grid Filter Example 1

This accepts a serial number in which the encoded number is a six-character string beginning with M followed by six numeric characters.

- **Filter**
  M[0-9]{6}
- **Effect**
  When a bar code, such as M270494, is scanned, all data is passed.

Grid Filter Example 2

This formats a scanned Social Security number and forms it into an XML element tagged “SSN”.

- **Filter**
  ([0-9]{3})([0-9]{2})([0-9]{4})= > <SSN > \1-\2-\3</SSN >
- **Effect**
  A bar code, such as 123456789, is passed and reformatted to <SSN > 123-45-6789</SSN >

Grid Filter Example 3

This deletes the first three and last five characters of a 21-character Code 128 label and deletes the first two characters of a 10-character Interleaved 2 of 5 label.

- **Filter**
  <]C > ...(.{13}).....= > \1; <]I > ..(.....)= > \1
- **Effect**
  If Code 128, AAA1234567890123BBBBB becomes 1234567890123
  If Interleaved 2 of 5, AA12345678 becomes 12345678

Grid Filter Example 4

This inverts data such that the first alphabetic string (like a first name) and second alphabetic string (like a last name) are reversed and separated by a comma and a space.

- **Filter**
  ([[:alpha:]])+ ([[:alpha:]])+= > \2, \1
- **Effect**
  When a bar code with the data "Dexter Gordon" is scanned, the data is modified to read "Gordon, Dexter".

ADC Connection

A 700 Series Computer can have both Bar Code and RFID reader engines with each engine supporting multiple connections. Each connection allows an application to access data or manage a configuration. An application could have multiple connections.

```
  // Get an instance of the ADC COM object that corresponds integrated scanner
IBarCodeReaderControl *pIBCControl;
  // Pointer to the Bar Code Reader object
HRESULT hrStatus = ITCDeviceOpen( TEXT("default"),
IID_IBarCodeReaderControl, ITC_DHDEVFLAG_READAHEAD,
(LPVOID *) &pIBCControl);
  // If the ADC object was successfully created and initialized, accept bar
code data.
ITC_BARCODE_DATA_DETAILS stBCDetails;
stBCDetails.wStructSize = sizeof(stBCDetails);
BYTE rgbBCData[1024];
  //Buffer used to accept the bar code data
DWORD dwBytesReceived;
  // Number of bytes in the return data.
HRESULT hrStatus = pIBCControl->Read(
rgbBCData,
sizeof(rgbBCData),
&dwBytesReceived,
& stBCDetails,
INFINITE
);
```

2D Imager Overview

The 700 Color optional integrated 2D Imager captures 640x480 256-grayscale images at 20 frames per second. The imager features can be categorized into data collection features and image acquisition features as follows:

Data Collection Features

The imager includes a decode engine capable of decoding 2D matrix symbologies such as Data Matrix as well as the traditional 1D and stacked symbologies *(see the table on the next page for supported symbologies)*. The application programming interfaces used to collect bar code data and configure the imager are the same as those used for the laser scanner. This includes the keyboard wedge as well as the ADC COM interfaces and includes functionality such as data editing and data filtering. In addition, the imager has the following configuration features (see "*IS9CConfig3 Functions*" starting on page 216 for configuration details):

- **Aimer LED**:
  A small, rectangular-aiming LED is displayed periodically during the image capture and decoding process. The initial duration *(after scan buttons are pressed)* of the aimer LED can be configured. This helps to select the specific bar code to be scanned with multiple bar codes in the image.
- **Scaled Illumination LED**:
  When the ambient light is not sufficient to decode the bar code, the red illumination LEDs will be turned on to brighten the image. The intensity of the illumination LEDs is scaled to brighten the image just enough for decode. The reduces power consumption and the effect of specular reflection.
- **Window size and position**:
  The default window size (640x480) can be reduced in size and positioned. This is useful in applications where multiple bar codes may be present in the image and the specific bar code must be selected to be read. For example, the window can be sized and positioned around the aimer LED. The entire bar code must reside in the configured window for a good decode.

Omni-directional scanning is a feature that does not require configuration. 1D and stacked symbologies as well as 2D matrix symbologies can be scanned with the 700 Series Computer in any orientation. Thus, time is not needed to orient the 700 horizontal as with laser scanners.

The following table shows which bar code symbologies are supported either by an imager or by a laser scanner.

Bar Code Symbology	Imager	Laser Scanner
Code 39	X	X
Interleaved 2 of 5	X	X
Standard 2 of 5	X	X
Matrix 2 of 5		X
Code 128	X	X
Code 93	X	X
Codabar	X	X
MSI		X
Plessey		X
UPC	X	X
EAN/EAN 128	X	X
Code 11		X
PDF 417	X	X
Micro PDF 417		X
Telepen		X
Data Matrix	X	
QR Code	X	

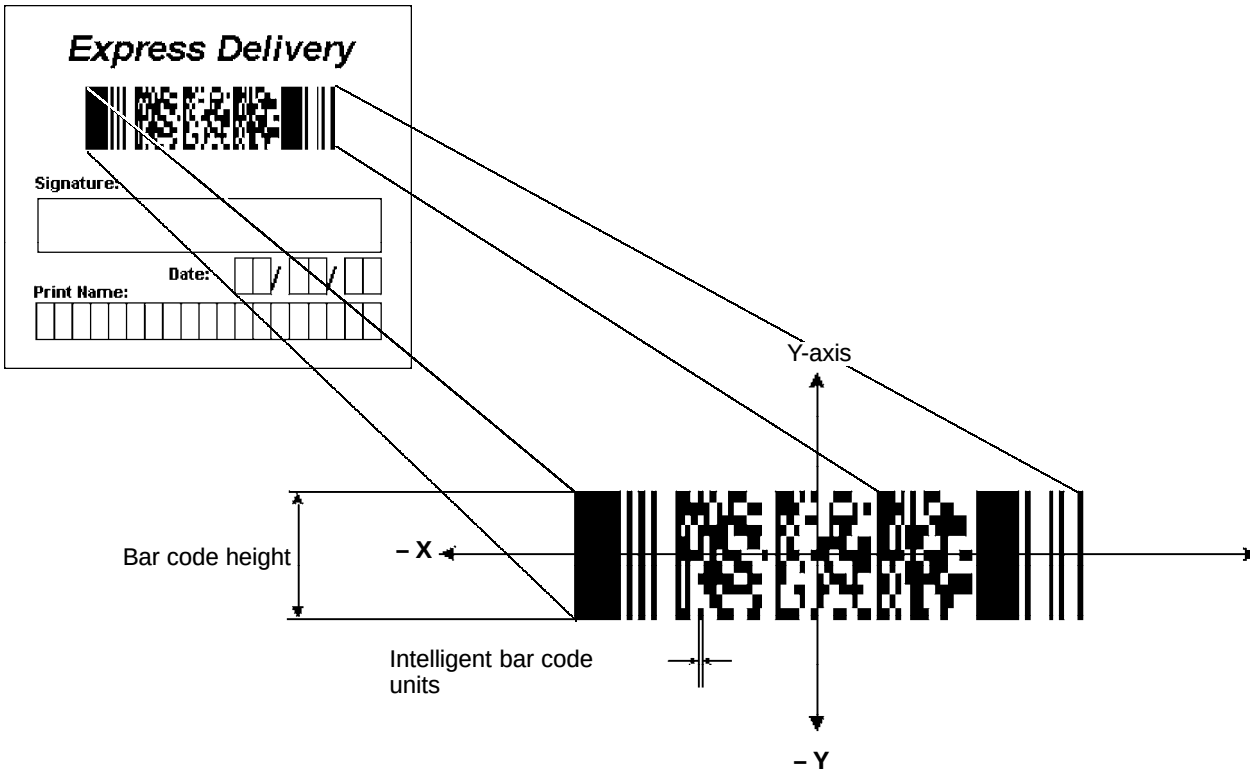
Image Acquisition Features

The integrated imager provides the following image acquisition features:

- **Real-time and Still Image Acquisition**:
  This includes functions that start and stop image acquisition and read acquired images.
- **Signature Capture**:
  This allows the application to retrieve an image of the normalized signature. This means the image is always oriented as if the picture were taken at right angles to the signature, at the same distance, and in the center of the image no matter in what orientation the picture was taken.

  Signature capture requires a PDF 417 or Code 128 bar code symbology to be present in the image and requires the application to identify the X,Y offsets relative to the center the bar code, the X,Y dimension of image to be captured, and the aspect ratio of the bar code. Note the units are in terms of the narrow element width of the bar code.

See the following example signature capture label and dimensions. These image acquisition features are provided through the IImage Interface defined on page 221.



Create and Delete ADC COM Object Functions

Use these functions to create and release ADC COM interfaces. ITCDEVMGMT.H is the header file and ITCDEVMGMT.LIB is the library.

ITCDeviceOpen

This function opens and initializes a communication channel to the device. In C++, this function returns a pointer to an interface on which the methods are called. In C, this function returns a handle, which is the first parameter in each of the interface function calls.

Syntax

```
HRESULT ITCDeviceOpen( LPCTSTR pszDeviceName, REFIID iid,
ITC_DEVICE_FLAGS eDeviceFlags, void** ppvObject );
```

Parameters

pszDevice	[in]	Pointer to a string that contains the device name on which to initialize the logical connection. The device name (Merlin 1) identifies a communications port. Use "default" for all internal scanners, such as Imager, SE900, etc. Use "ExtScanner" for tethered scanners.
iid	[in]	The identifier of the interface being requested.
eDeviceFlags	[in]	Enumeration that identifies the read characteristics as follows: • **ITC_DHDEVFLAG_READAHEAD** Data is buffered on behalf of the calling applications. Data Buffering starts after the first call to IADC::Read (). • **ITC_DHDEVFLAG_NODATA** The client application is managing the device to set its configuration or control its interface but not to collect data from the device.
ppvObject	[out]	A pointer to the interface pointer identified by *iid*. If the object does not support this interface, *ppvObject* is set to NULL.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

• ITCDeviceClose

ITCDeviceClose

This function closes the interface opened with ITCDeviceOpen.

Syntax:

```
HRESULT ITCDeviceClose( IUnknown** ppvObject );
```

Parameters

ppvObject	[in,out]	A pointer to the interface pointer created by ITCDeviceOpen. If successful on output, this pointer is set to NULL.

Return Values

None.

Remarks

On Windows, this interface decrements the reference count. So alternatively, IUnknown::Release() could be used and must be used if reference counting is performed with IUnknown::AddRef(). On DOS, this function closes all resources associated with the channel.

See Also

None.

IADC Functions

IADC functions provide ADC data in an input device independent manner. This interface can receive bar code data, RFID data, and other ADC data from data collection engines, such as a bar code scanner. Use IADC functions if bar code specifics such as symbology are not important to the application.

IADC functions are the following. IADC.H is the header file and ITCUUID.LIB contains the IID_IADC Interface GUID value used to obtain the interface.

- IADC::CancelReadRequest *(page 152)*
- IADC::Initialize *(page 153)*
- IADC::QueryAttribute *(page 154)*
- IADC::QueryData *(page 155)*
- IADC::Read *(page 156)*
- IADC::SetAttribute *(page 157)*

IADC::CancelReadRequest

This function cancels a pending Read() request. This call can be made on a separate thread as a Read() or on the same thread. On a separate thread, the function is useful in unblocking a blocked Read() so that other operations can be performed. On the same thread, this function is useful in stopping data from being collected on behalf of a Read Ahead Client.

Syntax

```
HRESULT IADC::CancelReadRequest( BOOL FlushBufferedData,WORD
*pwTotalDiscardedMessages, DWORD *pdwTotalDiscardedBytes );
```

Parameters

FlushBufferedData	[in]	True	Flush and discard all already buffered data.
		False	Do not discard data, data will be returned on the next read call.
pwTotalDiscardedMessages	[in/out]		Total number of discarded buffered labels or tags.
pdwTotalDiscardedBytes			Total number of discarded bytes.

Return Values

HRESULT that indicates success or failure.

Remarks

The return value indicates whether a read was pending.

See Also

- IADC::Initialize
- IADC::QueryAttribute
- IADC::QueryData
- IADC::Read
- IADC::SetAttribute

IADC::Initialize

This function initializes a connection by opening a communications channel with a logical reader engine. The communications port is implicitly identified. This communication channel is required to collect data or configure the device.

Syntax

```
HRESULT IADC::Initialize ( LPCTSTR pszDeviceName,
ITC_DEVICE_FLAGS eDeviceFlags ) ;
```

Parameters

pszDeviceName	[in]	Pointer to a string that contains the device name on which to initialize the logical connection. The device name (Merlin 1) identifies a communications port. Use "default" for all internal scanners, such as Imager, SE900, etc. Use "ExtScanner" for tethered scanners.
eDeviceFlags	[in]	Enumeration that identifies the read characteristic as follows: • **ITC_DHDEVFLAG_READAHEAD** Data is buffered on behalf of the calling application. Data buffering starts after the first call to IADC::Read ().

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

- IADC::CancelReadRequest
- IADC::QueryAttribute
- IADC::QueryData
- IADC::Read
- IADC::SetAttribute

IADC::QueryAttribute

This function retrieves a specified attribute that is device-independent. The specified attribute can be a grid or multiclient enable status.

Syntax

```
HRESULT IADC::QueryAttribute (
ITC_ADC_ATTRIBUTE_ID eAttribID, BYTE rgbBuffer[], DWORD
dwBufferSize, DWORD *pnBufferData );
```

Parameters

eAttribID	[in]	Specifies the attribute. Only one attribute can be queried at a time. See IADC::SetAttribute.
rgbBuffer	[out]	Contains buffer for the attribute to be queried. The structure of *lpBuffer* depends on the attribute being queried. See IADC::SetAttribute for a description of these structures.
dwBufferSize	[in]	The maximum number of bytes *rgbBuffer* can store.
pnBufferData	[out]	Pointer to the DWORD location to put the number of bytes stored in *rgbBuffer*.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

- IADC::CancelReadRequest
- IADC::Initialize
- IADC::QueryData
- IADC::Read
- IADC::SetAttribute

IADC::QueryData

This function returns the status of user input data that has been buffered.

Syntax

```
HRESULT IADC::QueryData ( DWORD *dwTotalBufferedBytes, WORD
*wNumberOfMessages, DWORD *dwNextMessageSize );
```

Parameters

dwTotalBufferedBytes	[out]	Total bytes buffered for connection.
wNumberOfMessages	[out]	Total messages buffered. For example, each buffer contains a single bar code scan.
dwNextMessageSize	[out]	Size (in bytes) of the next buffered message.

Return Values

A standard status code that indicates success or failure.

Remarks

None.

See Also

- IADC::CancelReadRequest
- IADC::Initialize
- IADC::QueryAttribute
- IADC::Read
- IADC::SetAttribute

IADC::Read

This function requests user input data from the reader engine. This is a blocking function that returns either when there is data or after a timeout.

Syntax

```
HRESULT IADC::Read ( BYTE rgbDataBuffer[], DWORD
dwDataBufferSize, DWORD pnBytesReturned, SYSTEMTIME
pSystemTime, DWORD dwTimeout );
```

Parameters

rgbDataBuffer	[in]	Pointer to the buffer that receives the data from the device.
dwDataBufferSize	[in]	Maximum number of bytes that can be stored in *rgbDataBuffer*.
pnBytesReturned	[out]	Pointer to the DWORD location to store the number of bytes returned in *rgbDataBuffer*.
pSystemTime	[out]	Pointer to a SYSTEMTIME structure that will hold the time stamp of the received data. This can be NULL if a timestamp is not needed.
dwTimeout	[in]	Number of milliseconds caller waits for data. This parameter is ignored if the Read Ahead flag is not set. • **0** If data is not available, returns quickly. • **INFINITE** Waits until data is available.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

- IADC::CancelReadRequest
- IADC::Initialize
- IADC::QueryAttribute
- IADC::QueryData
- IADC::SetAttribute

IADC::SetAttribute

This function changes an attribute such as a grid specification.

Syntax

```
HRESULT IADC::SetAttribute ( ITC_ADC_ATTRIBUTE_ID eAttribID,
BYTE rgbData[], DWORD nBufferSize );
```

Parameters

eAttribID [in] Identifies the attribute to set. Only one attribute can be set at a time. The attribute is:

- **ITC_MULTICLIENT_ENABLE**
  Indicates whether this client can coexist with other clients.

rgbData [in] Contains data for the attribute to be set. Depending on the *eAttribID*, this will be mapped to the appropriate structure as follows:

- **ITC_MULTICLIENT_ENABLE**
  BOOL is the *rgbData* Data Structure.
  - TRUE, Client can receive data with other clients *(default)*.
  - FALSE, Data stream to this client is turned off when there are other clients.
- **ITC_DHATTR_READFILTER**
  ITC_READFILTER is the *rgbData* Data Structure. The ITC_READFILE structure is defined in IADCDEVICE.H as follows:

```
typedef struct
     {
       #define ITC_MAXFILTER_CHARS 240
       WORD   nFilterChars;
       TCHAR    szFilter[ITC_MAXFILTER_CHARS];
     } ITC_READFILTER;
```

where:

- **ITC_MAXFILTER_CHARS**
  Maximum number of characters in a filter specification. Includes NULL termination.
  - *nFilterChars* Number of characters in *pszDataMask*.
  - *szFilter* Data mask specification. See "*Grid Data Filtering*."

nBufferSize [in] Number of bytes in *rgbData*.

ITC_DHATTR_READFILTER
Regular expression that performs data filtering and data editing. See "*Grid Data Filtering*" on page 141 for more information.

Return Values

A standard status code that indicates success or failure.

Remarks

None.

See Also

- IADC::CancelReadRequest
- IADC::Initialize
- IADC::QueryAttribute
- IADC::QueryData
- IADC::Read

IBarCodeReaderControl Functions

IBarCodeReaderControl functions provide functionality for bar code collection and control only. These functions allow an application to:

- Trigger the bar code laser scanner
- Disable the scanner
- Receive a bar code with details such as symbology scanned, data type (Unicode, ASCII), and the time the data was received.

These functions include the following. IBARCODEREADER.H is the header file and ITCUUID.LIB contains the IID_IADC Interface GUID value used to obtain the interface.

IBarCodeReaderControl::CancelReadRequest

This function cancels a pending IBarCodeReaderControl::Read request. If the read request is blocked, issue the CancelReadRequest from a separate thread.

Syntax

```
HRESULT IBarCodeReaderControl::CancelReadRequest( BOOL
FlushBufferedData, WORD *pwTotalDiscardedMessages,WORD
*pwTotalDiscardedBytes );
```

Parameters

FlushBufferedData	[in]	TRUE	Flushes and discards all buffered data.
		FALSE	Does not discard data; data will be returned on the next read call.
pwTotalDiscardedMessages	[in/out]		Total number of discarded buffered labels or tags.
pwTotalDiscardedBytes			Total number of discarded bytes.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

- IBarCodeReaderControl::ControlLED
- IBarCodeReaderControl::Initialize
- IBarCodeReaderControl::IssueBeep
- IBarCodeReaderControl::QueryAttribute
- IBarCodeReaderControl::Read
- IBarCodeReaderControl::SetAttribute
- IBarCodeReaderControl::TriggerScanner

IBarCodeReaderControl::ControlLED

This function controls LED illumination on a tethered scanner. The good read LED and any valid LEDs will be turned on and off based on defined parameters.

Syntax

```
HRESULT IBarCodeReaderControl::ControlLED(
ITC_BARCODE_LASER_LED_ID eLED, BOOL fLedOn );
```

Parameters

eLED [in] The specified LED identifier.
- **ITC_BARCODE_LASER_GOOD_READ_LED** Identifies the good read LED.

fLedOn [in] TRUE turns on the LED. FALSE turns off the LED.

Return Values

HRESULT that indicates success or failure.

Remarks

This function does not coordinate LED control with the scanner. If the scanner LED control is enabled, function results will be unpredictable.

See Also

- IBarCodeReaderControl::CancelReadRequest
- IBarCodeReaderControl::Initialize
- IBarCodeReaderControl::IssueBeep
- IBarCodeReaderControl::QueryAttribute
- IBarCodeReaderControl::Read
- IBarCodeReaderControl::SetAttribute
- IBarCodeReaderControl::TriggerScanner

IBarCodeReaderControl::Initialize

This function opens and initializes a communications channel with a logical bar code reader engine.

Syntax

```
HRESULT IBarCodeReaderControl::Initialize ( LPCTSTR
pszDeviceName, ITC_DEVICE_FLAGS eDeviceFlags ) ;
```

Parameters

pszDeviceName [in] Pointer to a string with device on which to initialize the logical connection. The device identifies a communications port. Use "default" for all internal scanners, such as Imager, SE900, etc. Use "ExtScanner" for tethered scanners.

eDeviceFlags [in] Enumeration that identifies the read characteristic as follows:

- **ITC_DHDEVFLAG_READAHEAD** Data is buffered on behalf of the calling applications. Data Buffering starts after the first call to IADC::Read ().

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

- IBarCodeReaderControl::CancelReadRequest
- IBarCodeReaderControl::ControlLED
- IBarCodeReaderControl::IssueBeep
- IBarCodeReaderControl::QueryAttribute
- IBarCodeReaderControl::Read
- IBarCodeReaderControl::SetAttribute
- IBarCodeReaderControl::TriggerScanner

IBarCodeReaderControl::IssueBeep

This function causes the reader engine to generate a high beep, a low beep, or a custom beep. The high beep and low beep are preconfigured beep tones and durations. The custom beep allows the client to specify the frequency and duration. The volume is the current volume setting. *Note this is not implemented.*

Syntax

```
HRESULT IBarCodeReaderControl::IssueBeep( ITC_BEEP_SPEC
rgBeepRequests[], DWORD dwNumberOfBeeps );
```

Parameters

rgBeepRequests [in] Array of ITC_BEEP_SPEC structures that identifies the beep type. The beep structure is:

```
typedef struct tagITCBeepSpec
{
ITC_BEEP_TYPE eBeepType; // Identifies the type of beep
// Following fields used only if the beep type is ITC_CUSTOM_BEEP.
WORD wPitch; // Frequency, in Hz, of the beep.
WORD wOnDuration; // Duration, in milliseconds, of Beep On.
WORD wOffDuration; // Duration, in milliseconds, of Beep Off
// Beep Off is used to separate individual beeps
} ITC_BEEP_SPEC;
typedef enum tagITCBeepType
{
ITC_LOW_BEEP, // Issue the default low beep.
ITC_HIGH_BEEP, // Issue the default high beep.
ITC_CUSTOM_BEEP, // Issue a custom beep.
} ITC_BEEP_TYPE;
```

dwNumberOfBeeps [in] Identifies the total number of beeps in *rgBeepRequests*.

Return Values

E_NOTIMPL as this function is not implemented.

Remarks

None.

See Also

- IBarCodeReaderControl::CancelReadRequest
- IBarCodeReaderControl::ControlLED
- IBarCodeReaderControl::Initialize
- IBarCodeReaderControl::QueryAttribute
- IBarCodeReaderControl::Read
- IBarCodeReaderControl::SetAttribute
- IBarCodeReaderControl::TriggerScanner

IBarCodeReaderControl::QueryAttribute

This function retrieves the device-specific grid, the scanner enable status, and the LED control status for the current bar code reader engine.

Syntax

```
HRESULT IBarCodeReaderControl::QueryAttribute (
ITC_BARCODEREADER_ATTRIBUTE_ID eAttr, BYTE rgbAttrBuffer[],
DWORD dwAttrBufferSize );
```

Parameters

eAttr	[in]	Specifies the attribute. See IBarCodeReaderControl::SetAttribute on page 167 for the attributes.
rgbAttrBuffer	[out]	Contains buffer for the attribute to be queried. The structure of *rgbAttrBuffer* depends on the attribute being queried. See IBarCodeReaderControl::SetAttritbute for a description of these structures.
dwAttrBufferSize	[in]	The maximum number of bytes that *rgbAttrBuffer* can store.

Return Values

A standard status code that indicates success or failure.

Remarks

The following attributes are not supported on the imager:

- ITC_RDRATTR_TONE_ENABLE
- ITC_RDRATTR_VOLUME_LEVEL
- ITC_RDRATTR_TONE_FREQUENCY
- ITC_RDRATTR_GOOD_READ_BEEPS_NUMBER
- ITC_RDRATTR_GOOD_READ_BEEP_DURATION

See Also

- IBarCodeReaderControl::CancelReadRequest
- IBarCodeReaderControl::ControlLED
- IBarCodeReaderControl::Initialize
- IBarCodeReaderControl::IssueBeep
- IBarCodeReaderControl::Read
- IBarCodeReaderControl::SetAttribute
- IBarCodeReaderControl::TriggerScanner

IBarCodeReaderControl::Read

This function reads data from the bar code input device. This method performs the same function as IADC::Read () except that it provides additional information about data received such as bar code symbology used, data type, and time stamp of received data.

Syntax

```
HRESULT IBarCodeReaderControl::Read ( BYTE
rgbDataBuffer[],DWORD dwDataBufferSize, DWORD
pnBytesReturned,ITC_BARCODE_DATA_DETAILS
pBarCodeDataDetails, DWORD dwTimeout );
```

Parameters

rgbDataBuffer	[in]	Pointer to the buffer that receives data from the device.
dwDataBufferSize	[in]	Maximum number of bytes that can be stored in *rgbDataBuffer*.
pnBytesReturned	[out]	Pointer to the DWORD location that will store the bytes returned in *rgbDataBuffer*.
pBarCodeDataDetails	[in]	Address of data structure in which to put the data details. This may be NULL. The ITC_BARCODE_DATA_DETAILS is:

```
typedef struct tagITCBarCodeDetails

{

WORD wStructSize,
ITC_BARCODE_SYMBOLOGY_ID eSymbology,
ITC_BARCODE_DATATYPE eDataType,
SYSTEMTIME stTimeStamp,
}ITC_BARCODE_DATA_DETAILS;

typedef enum tagBarCodeDataType

{

BARCODE_DATA_TYPE_UNKNOWN = 1,
BARCODE_DATA_TYPE_ASCII,
BARCODE_DATA_TYPE_UNICODE,
}ITC_BARCODE_DATATYPE;
```

where:

- *wStructSize* Size of structure. Used for versioning structure.
- *eSymbology* Symbology of the returned data.
- *eDataType* Identifies data types as ASCII, UNICODE, etc.
- *stTimeStamp* Timestamp of the received data.
- **BARCODE_DATA_TYPE_UNKNOWN** Data in unknown.
- **BARCODE_DATA_TYPE_ASCII** Data is ASCII.
- **BARCODE_DATA_TYPE_UNICODE** Data is UNICODE.

dwTimeout	[in]	Number of milliseconds caller waits for data. If you set a timeout, the call will be blocked until data is received. • **0** If data not available, returns quickly. • **INFINITE** Waits until data is available.

Return Values
HRESULT that indicates success or failure.

Remarks
None.

See Also
- IBarCodeReaderControl::CancelReadRequest
- IBarCodeReaderControl::ControlLED
- IBarCodeReaderControl::Initialize
- IBarCodeReaderControl::IssueBeep
- IBarCodeReaderControl::QueryAttribute
- IBarCodeReaderControl::SetAttribute
- IBarCodeReaderControl::TriggerScanner

IBarCodeReaderControl::SetAttribute

This function enables and disables the laser scanner, sets the bar code reader engine specific grid, and enables or disables the reader engine LED control.

Syntax

```
HRESULT IBarCodeReaderControl::SetAttribute (
ITC_BARCODEREADER_ATTRIBUTE_ID eAttr, BYTE rgbAttrBuffer[],
DWORD dwAttrBufferSize );
```

Parameters

eAttr [in] Identifies the attribute to set. Only one attribute can be set at a time. The attributes are:

- **ITC_RDRATTR_SCANNER_ENABLE**
  Enable or disable scanner for all connections.
- **ITC_RDRATTR_GOOD_READ_LED_ENABLE**
  Enables and disables the reader engine from controlling the good read LED.
- **ITC_RDRATTR_TONE_ENABLE**
  Enables and disables the reader engine from issuing beeps.
- **ITC_RDRATTR_VOLUME_LEVEL**
  Sets beep volume level.
- **ITC_RDRATTR_TONE_FREQUENCY**
  Sets beep frequency.
- **ITC_RDRATTR_GOOD_READ_BEEPS_NUMBER**
  Sets number of beeps for a good read.
- **ITC_RDRATTR_GOOD_READ_BEEP_DURATION**
  Sets duration of beeps for a good read.
- **ITC_DHATTR_READFILTER**
  The ITC_READFILTER is the *rgbData* Data Structure. The ITC_READFILE structure is defined in IADCDEVICE.H as follows:

```
typedef struct
     {
       #define ITC_MAXFILTER_CHARS 240
       WORD nFilterChars;
       TCHAR szFilter[ITC_MAXFILTER_CHARS];
     } ITC_READFILTER;
```

where:

- *nFilterChars* Number of characters in *pszDataMask*.
- *szFilter* Data mask specification. See "*Grid Data Filtering*."
- **ITC_MAXFILTER_CHARS**
  Maximum number of characters in a filter specification. Includes NULL termination.

rgbAttrBuffer [in] Contains data for the attribute to be set. Depending on *eAttr*, the *rgbAttrData* will be mapped to the appropriate structure as shown in the following table .

rgbAttrBuffer Data Structures

eAttr	Data Structure contained in rgbAttrBuffer
ITC_RDRATTR_GRID	ITC_BARCODEREADER_READER_GRID Reader Engine specific grid only.
ITC_RDRATTR_SCANNER_ENABLE	BOOL TRUE Enable scanner. FALSE Disable scanner.
ITC_RDRATTR_GOOD_READ_LED_ENABLE	BOOL TRUE Reader Engine controls good read LED. FALSE Good read LED is not controlled.
ITC_RDRATTR_DATA_VALID_LED_ENABLE	BOOL TRUE Reader Engine controls data valid LED. FALSE Data valid LED is not controlled.
ITC_RDRATTR_TONE_ENABLE	BOOL TRUE Reader Engine issues beeps. FALSE Beeps are not issued.

ITC_RDRATTR_VOLUME_LEVEL	ITC_BEEP_VOLUME
	An enumerator that identifies the beep volume level control. Valid range for S9C:
	`typedef enum tagBeepVolume`
	`{`
	`ITC_BEEP_VOLUME_LOW = 0,`
	`ITC_BEEP_VOLUME_MEDIUM = 2,`
	`ITC_BEEP_VOLUME_HIGH = 1 //Default`
	`}ITC_BEEP_VOLUME`
	***Note**: Due to the hardware design on this 700 Series Computer, the volume level can be either OFF (ITC_BEEP_VOLUME_LOW) or ON (ITC_BEEP_VOLUME_MEDIUM/HIGH).*

ITC_RDRATTR_TONE_FREQUENCY	DWORD A value that identifies the tone frequency in Hz. Valid range for S9C: 1000`4095 Hz *(default: 2090).* ***Note**: Value is divided by 10 for storage. On retrieval, the scanner rounds off the value to the nearest 10 Hz, then multiplies the value by 10. For example, the value sent to the scanner is 2095. On retrieval, the value returned is 2090.*

rgbAttrBuffer Data Structures (continued)

eAttr	Data Structure contained in rgbAttrBuffer

ITC_RDRATTR_GOOD_READ_BEEPS_NUMBER	ITC_GOOD_READ_BEEPS_NUMBER
	An enumerator identifying the good read beeps number.
	Valid range for S9C:
	`typedef enum tagGoodReadBeepsNumber`
	`{`
	`ITC_NUM_BEEPS_NONE = 0,`
	`ITC_NUM_BEEPS_ONE = 1, // Default`
	`ITC_NUM_BEEPS_TWO = 2`
	`}ITC_GOOD_READ_BEEPS_NUMBER`

ITC_RDRATTR_GOOD_READ_BEEP_DURATION	DWORD Value identifying the good read beep duration in ms. Valid range for S9C: 0`2550 ms *(Default: 80)*. ***Note**: Value is divided by 10 for storage. On retrieval, the scanner rounds the value to the nearest 10 ms, then multiplies the value by 10.*

dwAttrBufferSize [in] The size of *rgbAttrBuffer* in bytes.

Return Values

HRESULT that indicates success or failure.

Remarks

Read ahead and non-read ahead clients can change the grid. Since changing the grid changes the entire reader engine grid, use IBarCodeReaderControl::QueryAttribute to retrieve the current reader engine grid and grid changes before sending back using SetAttribute. The grid structure is *typedef struct tagBarCodeReaderGrid*.

```
{
ITC_DI_GRID stDIGrid; // Device independent grid.
ITC_DDBARCODE_GRID stDDGrid; // Reader engine dependent grid
DWORD dwDataSourceTypeMask;
} ITC_BARCODEREADER_GRID;

ITC_DI_GRID

typedef struct tagItcBarCodeGrid
{
DWORD dwSymbologyMask; // Symbologies to be received.
} ITC_DDBARCODE_GRID;
```

When the scanner is enabled, it scans when the scan button is pressed or the trigger is pulled. When the scanner is disabled, it does not respond when the scan button is pressed or the trigger is pulled.

The following attributes are not supported on the imager:

- ITC_RDRATTR_TONE_ENABLE
- ITC_RDRATTR_VOLUME_LEVEL
- ITC_RDRATTR_TONE_FREQUENCY
- ITC_RDRATTR_GOOD_READ_BEEPS_NUMBER
- ITC_RDRATTR_GOOD_READ_BEEP_DURATION

See Also

- IBarCodeReaderControl::CancelReadRequest
- IBarCodeReaderControl::ControlLED
- IBarCodeReaderControl::Initialize
- IBarCodeReaderControl::IssueBeep
- IBarCodeReaderControl::QueryAttribute
- IBarCodeReaderControl::Read
- IBarCodeReaderControl::TriggerScanner

IBarCodeReaderControl::TriggerScanner

This function turns the scanner on and off. The client application must coordinate control of the scanner with the user.

Syntax

```
HRESULT IBarCodeReaderControl::TriggerScanner ( BOOL
fScannerOn );
```

Parameters

fScannerOn [in] Set TRUE to turn the scanner on. Set FALSE to turn the scanner off.

Return Values

HRESULT that indicates success or failure.

Remarks

The scanner will be turned on or off independent of the actions of the users. The client application must coordinate control of the scanner with the user. When the scanner is turned on, its behavior is controlled by the trigger mode. That is, in one shot mode, the laser turns off when a label is scanned; in auto-trigger mode, the laser remains on.

See Also

- IBarCodeReaderControl::CancelReadRequest
- IBarCodeReaderControl::ControlLED
- IBarCodeReaderControl::Initialize
- IBarCodeReaderControl::IssueBeep
- IBarCodeReaderControl::QueryAttribute
- IBarCodeReaderControl::Read
- IBarCodeReaderControl::SetAttribute

IS9CConfig Functions

This interface provides methods to set and retrieve the 700 Series Computer bar code configuration. All supported symbologies are initialized to their defaults when the S9C firmware is loaded.

GET/SET functions use enumerations as their parameters. In most enumerations, there is an enumerator xx_NO_CHANGE (such as ITC_CODE39_NO_CHANGE), where xx refers to a particular enumeration. This enumerator can be used during a call to a SET to indicate that no change is to be made to that particular parameter. This prevents the called function from having to format the same S9C command and send down to the S9C scanner.

For all symbologies, to set a bar code length of "any length," use a value of "0" for the bar code length argument.

IS9CConfig functions are the following. IS9CCONFIG.H is the header file and ITCUUID.LIB contains the IID_IADC Interface GUID value used to obtain the interface.

IS9CConfig::GetCodabar

This function retrieves the current settings of Codabar symbology.

Syntax

```
HRESULT IS9CConfig::GetCodabar( ITC_CODABAR_DECODING*
peDecode, ITC_CODABAR_START_STOP* peSS, ITC_CODABAR_CLSI*
peCLSI, ITC_CODABAR_CHECK_DIGIT* peCheck,
ITC_BARCODE_LENGTH_ID* peLengthId, BYTE rgbLengthBuff[],
DWORD* pdwNumBytes );
```

Parameters

peDecode	[out]	Pointer to the ITC_CODABAR_DECODING location to receive the decoding for Codabar symbology.
peSS	[out]	Pointer to the ITC_CODABAR_START_STOP location to receive the Start/Stop option.
peCLSI	[out]	Pointer to the ITC_CODABAR_CLSI location to receive the CLSI library system.
peCheck	[out]	Pointer to the ITC_CODABAR_CHECK_DIGIT location to receive the check digit.
peLengthId	[out]	Pointer to the ITC_BARCODE_LENGTH_ID location to receive an indicator of either ITC_BARCODE_LENGTH or ITC_BARCODE_FIXED_LENGTH.
rgbLengthBuff	[out,size_is(3)]	An array of bytes to receive 1 byte of data for ITC_BARCODE_LENGTH, or 3 bytes of data for ITC_BARCODE_FIXED_LENGTH.
pdwNumBytes	[out]	Pointer to the DWORD location to receive a number indicating number of bytes in *rbgLengthBuff[]*: 1 byte for ITC_BARCODE_LENGTH or 3 bytes for ITC_BARCODE_FIXED_LENGTH.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

None.

IS9CConfig::SetCodabar

This function updates the Codabar settings with new values.

Syntax

```
HRESULT IS9CConfig::SetCodabar( ITC_CODABAR_DECODING
eDecode, ITC_CODABAR_START_STOP eSS, ITC_CODABAR_CLSI
eCLSI,ITC_CODABAR_CHECK_DIGIT eCheck, ITC_BARCODE_LENGTH_ID
eLengthId, BYTE rgbLengthBuff[],DWORD dwNumBytes );
```

Parameters

eDecode	[in]	Identifies the decoding for Codabar symbology.
eSS	[in]	Identifies the Start/Stop option.
eCLSI	[in]	Identifies the CLSI library system.
eCheck	[in]	Identifies the check digit.
eLengthId	[in]	Use ITC_BARCODE_LENGTH_NO_CHANGE to indicate no change for bar code length. Use ITC_BARCODE_LENGTH for any length and minimum length, and set *rgbLengthBuff[0]* to a valid length value. Use ITC_BARCODE_FIXED_LENGTH to compose 1 or 2 or 3 fixed lengths, and set 3 bytes: *rgbLengthBuff[0]*, *rgbLengthBuff[1]*, *rgbLengthBuff[2]* with valid values.
rgbLengthBuff	[in,size_is(dwNumBytes)]	An array of bytes containing bar code lengths when *eLengthId* = ITC_BARCODE_LENGTH or ITC_BARCODE_FIXED_LENGTH.
dwNumBytes	[in]	Number of bytes in *rbgLengthBuff[]*. For S9C, this value is 1 when *eLengthId* = ITC_BARCODE_LENGTH or 3 when *eLengthId* = ITC_BARCODE_FIXED_LENGTH

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

None.

Codabar Default Settings

Parameter	Default	Valid Range
Decode	Not Active	ITC_CODABAR_DECODING
CLSI Library System	Not Active	ITC_CODABAR_CLSI
Start/Stop	Not Transmitted	ITC_CODABAR_START_STOP
Check Digit	Not Used	ITC_CODABAR_CHECK_DIGIT
Bar Code Length	Minimum Length = 6	0x00`0xFE ITC_BC_LENGTH_NO_CHANGE

Codabar Enumerations

```
typedef enum tagCodabarDecoding
{
ITC_CODABAR_NOTACTIVE = 0,           // Default
ITC_CODABAR_ACTIVE = 1,
ITC_CODABAR_NO_CHANGE = 255
} ITC_CODABAR_DECODING;

typedef enum tagCodabarStartStop
{
ITC_CODABAR_SS_NOTXMIT,           // Default
ITC_CODABAR_SS_LOWERABCD,         // a,b,c,d
ITC_CODABAR_SS_UPPERABCD,         // A,B,C,D
ITC_CODABAR_SS_LOWERABCDTN,       // a,b,c,d / t,n,*,e
ITC_CODABAR_SS_DC1TODC4,          // DC1,DC2,DC3,DC4
ITC_CODABAR_SS_NO_CHANGE = 255
} ITC_CODABAR_START_STOP;

typedef enum tagCodabarClsi
{
ITC_CODABAR_CLSI_NOTACTIVE = 0,   // Default
ITC_CODABAR_CLSI_ACTIVE = 1,
ITC_CODABAR_CLSI_NO_CHANGE = 255
} ITC_CODABAR_CLSI;

typedef enum tagCodabarCheckDigit
{
ITC_CODABAR_CHECK_NOTUSED,           // Default
ITC_CODABAR_CHECK_XMIT,
ITC_CODABAR_CHECK_NOTXMIT,
ITC_CODABAR_CHECK_NO_CHANGE = 255
} ITC_CODABAR_CHECK_DIGIT;

typedef enum tagBarcodeLengthId
{
ITC_BARCODE_LENGTH = 0,
ITC_BARCODE_FIXED_LENGTH,
ITC_BARCODE_LENGTH_NO_CHANGE = 255
} ITC_BARCODE_LENGTH_ID;
```

IS9CConfig::GetCode39

This function retrieves the current settings of Code 39.

Syntax

```
HRESULT IS9Cconfig::GetCode39( ITC_CODE39_DECODING*
peDecode, ITC_CODE39_FORMAT* peFormat,
ITC_CODE39_START_STOP* peSS,ITC_CODE39_SS_CHARS* peSSChars,
ITC_CODE39_CHECK_DIGIT* peCheck, DWORD* pwLength );
```

Parameters

peDecode	[out]	Pointer to the ITC_CODE39_DECODING location to receive the decoding for Code 39.
peFormat	[out]	Pointer to the ITC_CODE39_FORMAT location to receive the Code 39 format.
peSS	[out]	Pointer to the ITC_CODE39_START_STOP location to receive the Code 39 start/stop.
peSSChars	[out]	Pointer to the ITC_CODE39_SS_CHARS location to receive the Start/Stop character.
peCheck	[out]	Pointer to the ITC_CODE39_CHECK_DIGIT location to receive the check digit.
pwLength	[out]	Pointer to the DWORD location to receive the bar code length.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

None.

IS9CConfig::SetCode39

This function updates the Code 39 settings with new values.

Syntax

```
HRESULT IS9CConfig::SetCode39( ITC_CODE39_DECODING
eDecode,ITC_CODE39_FORMAT eFormat, ITC_CODE39_START_STOP
eSS,ITC_CODE39_SS_CHARS eSSChars, ITC_CODE39_CHECK_DIGIT
eCheck, DWORD dwLength );
```

Parameters

eDecode	[in]	Identifies the decoding for Code 39.
eFormat	[in]	Identifies the Code 39 Format.
eSS	[in]	Identifies the Start/Stop option.
eSSChars	[in]	Identifies the Start/Stop character.
eCheck	[in]	Identifies the Check digit.
dwLength	[in]	Identifies the bar code length.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

None.

Code 39 Default Settings

Parameter	Default	Valid Range
Decoding	Active	ITC_CODE39_DECODING
Format	Standard 43 Character	ITC_CODE39_FORMAT
Start/Stop	Not Transmitted	ITC_CODE39_START_STOP
Accepted Start/stop Characters	* only	ITC_CODE39_SS_CHARS
Check Digit	Not Used	ITC_CODE39_CHECK_DIGIT
Bar Code Length	Any Bar Code Length	0x00`0xFE ITC_BC_LENGTH_NO_CHANGE

Code 39 Enumerations

```
typedef enum tagCode39Decoding
{
ITC_CODE39_NOTACTIVE = 0,
ITC_CODE39_ACTIVE = 1,            // Default
ITC_CODE39_NO_CHANGE = 255
} ITC_CODE39_DECODING;

typedef enum tagCode39Format
{
ITC_CODE39_FORMAT_STANDARD43,         // Default
ITC_CODE39_FORMAT_FULLASCII,
ITC_CODE39_FORMAT_NO_CHANGE = 255
} ITC_CODE39_FORMAT;

typedef enum tagCode39StartStop
{
ITC_CODE39_SS_NOTXMIT,            // Default
ITC_CODE39_SS_XMIT,
ITC_CODE39_SS_NO_CHANGE = 255
} ITC_CODE39_START_STOP;

typedef enum tagCode39StartStopChars
{
ITC_CODE39_SS_CHARS_DOLLARSIGN,
ITC_CODE39_SS_CHARS_ASTERISK,         // Default
ITC_CODE39_SS_CHARS_BOTH,
ITC_CODE39_SS_CHARS_NO_CHANGE = 255
} ITC_CODE39_SS_CHARS;

typedef enum tagCode39CheckDigit
{
ITC_CODE39_CHECK_NOTUSED,         // Default
ITC_CODE39_CHECK_MOD43_XMIT,
ITC_CODE39_CHECK_MOD43_NOTXMIT,
ITC_CODE39_CHECK_FRENCH_CIP_XMIT,
ITC_CODE39_CHECK_FRENCH_CIP_NOTXMIT,
ITC_CODE39_CHECK_ITALIAN_CPI_XMIT,
ITC_CODE39_CHECK_ITALIAN_CPI_NOTXMIT,
ITC_CODE39_CHECK_NO_CHANGE = 255
} ITC_CODE39_CHECK_DIGIT;
```

IS9CConfig::GetCode93

This function retrieves the current settings of Code 93.

Syntax

```
HRESULT IS9CConfig::GetCode93( ITC_CODE93_DECODING*
peDecode, DWORD* pdwLength );
```

Parameters

peDecode	[out]	Pointer to the ITC_CODE93_DECODING location to receive the decoding for Code 93 symbology.
pdwLength	[out]	Pointer to the DWORD location to receive a value for bar code length.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

None.

IS9CConfig::SetCode93

This function updates the Code 93 settings with new values.

Syntax

```
HRESULT IS9CConfig::SetCode93( ITC_CODE93_DECODING
eDecode,DWORD dwLength );
```

Parameters

eDecode	[in]	Identifies the decoding for Code93 Symbology.
dwLength	[in]	Identifies the bar code length.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

None.

Code 93 Default Settings

Parameter	Default	Valid Range
Decoding	Not Active	ITC_CODE93_DECODING
Bar Code Length	Any Bar Code Length	0x00`0xFE ITC_BC_LENGTH_NO_CHANGE

Code 93 Enumerations

Use this when the bar code length does not require any change.

```
typedef enum tagCode93Decoding
{
ITC_CODE93_NOTACTIVE = 0,      // Default
ITC_CODE93_ACTIVE = 1,
ITC_CODE93_NO_CHANGE = 255
} ITC_CODE93_DECODING;
#define ITC_BC_LENGTH_NO_CHANGE 255.
```

IS9CConfig::GetCode128

This function retrieves the current settings of Code 128 symbology.

Syntax

HRESULT **IS9Cconfig::GetCode128(** ITC_CODE128_DECODING* *peDecode*, ITC_EAN128_IDENTIFIER* *peEan128Ident*, ITC_CODE128_CIP128 *peCip128State*, BYTE* *pbyFNC1*, DWORD* *pdwLength* **);**

Parameters

peDecode	[out]	Pointer to the ITC_CODE128_DECODING location to receive the decoding for Code 128 symbology.
peEan128Ident	[out]	Pointer to the ITC_EAN128_IDENTIFIER location to receive the EAN 128 identifier.
peCip128State	[out]	Pointer to the ITC_CODE128_CIP128 location to receive the CIP 128.
pbyFNC1	[out]	Pointer to the BYTE location to receive the FNC1 separator character.
pdwLength	[out]	Pointer to the DWORD location to receive a value for bar code length.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

None.

IS9CConfig::SetCode128

This function updates the Code 128 settings with new values.

Syntax

```
HRESULT IS9CConfig::SetCode128( ITC_CODE128_DECODING
eDecode, ITC_EAN128_IDENTIFIER eEan128Ident,
ITC_CODE128_CIP128 eCip128State, BYTE byFNC1, DWORD dwLength
);
```

Parameters

eDecode	[in]	Identifies the decoding for Code 128 symbology.
eEan128Ident	[in]	Identifies the EAN 128 identifier.
eCip128State	[in]	Identifies the CIP 128.
byFNC1	[in]	Identifies the FNC1 separator character, usually any ASCII value.
dwLength	[in]	Identifies the bar code length.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

None.

Code 128/EAN 128 Default Settings

Parameter	Default	Valid Range	
Decoding	Not Active	ITC_CODE128_DECODING	
EAN 128 Identifier	Include]C1	ITC_EAN128_IDENTIFIER	
CIP 128 French Pharmaceutical Codes	Not Active	ITC_CODE128_CIP128	
FNC1 Separator Character (EAN 128 norms)	GS function Char ASCII 29 or 0x1D	0x00`0xFE	ITC_CODE128_FNC1_NO_CHANGE
Bar Code Length	Any Bar Code Length	0x00`0xFE	ITC_BC_LENGTH_NO_CHANGE

Code 128 Enumerations

```
typedef enum tagCode128Decoding
{
ITC_CODE128_NOTACTIVE = 0, // Default
ITC_CODE128_ACTIVE = 1,
ITC_CODE128_NO_CHANGE = 255
} ITC_CODE128_DECODING;
typedef enum tagEan128Identifier
{
ITC_EAN128_ID_REMOVE,
ITC_EAN128_ID_INCLUDE, // Default
ITC_EAN128_ID_NO_CHANGE = 255
} ITC_EAN128_IDENTIFIER;
typedef enum tagCode128Cip128
{
ITC_CODE128_CIP128_NOTACTIVE = 0, // Default
ITC_CODE128_CIP128_ACTIVE = 1,
ITC_CODE128_CIP128_NO_CHANGE = 255
} ITC_CODE128_CIP128;

#define ITC_CODE128_FNC1_NO_CHANGE 255.
This definition can be used when the Code128 FNC1 does not require any change.

#define ITC_BC_LENGTH_NO_CHANGE 255. This definition can be used when the bar
code length does not require any change.
```

The table below shows what to be expected for EAN 128 labels for various symbology identifier transmit configurations and EAN 128 Identifier options.

Setup			**Application's Expected Result**	
EAN 128]C1 ID		**Symbology ID option**	**EAN 128 Label**	**Other Labels**
1	Include]C1	Disabled	<data>	<data>
2	Remove]C1	Disabled	<data>	<data>
3	Include]C1	AIM ID Transmitted	]C1<data>	]XY<data>
4	Remove]C1	AID ID Transmitted	]C1<data>	]XY<data>
5	Include]C1	Custom ID Transmitted	Z]C1<data>	Z<data>
6	Remove]C1	Custom ID Transmitted	Z<data>	Z<data>
where "X" is the symbology identifier, "Y" is the modifier character, and "Z" is the 1-byte symbology identifier.				

IS9CConfig::GetI2of5

This function retrieves the current settings of Interleaved 2 of 5.

Syntax

```
HRESULT IS9CConfig::GetI2of5( ITC_INTERLEAVED2OF5_DECODING*
peDecode, ITC_INTERLEAVED2OF5_CHECK_DIGIT* peCheck,
ITC_BARCODE_LENGTH_ID* peLengthId, BYTE rbgLengthBuff[],
DWORD* pdwNumBytes );
```

Parameters

peDecode	[out]	Pointer to the ITC_INTERLEAVED2OF5_DECODING location to receive the decoding for Interleaved 2 of 5 symbology.
peCheck	[out]	Pointer to the ITC_INTERLEAVED2OF5_CHECK_DIGIT location to receive the check digit.
peLengthId	[out]	Pointer to the ITC_BARCODE_LENGTH_ID location to receive an indicator of either ITC_BARCODE_LENGTH or ITC_BARCODE_FIXED_LENGTH.
rgbLengthBuff	[out,size_is(3)]	An array of bytes to receives 1 byte of data for ITC_BARCODE_LENGTH or 3 bytes of data for ITC_BARCODE_FIXED_LENGTH.
pdwNumBytes	[out]	Pointer to the DWORD location to receive a number indicating number of bytes in *rbgLengthBuff[]*: 1 byte for ITC_BARCODE_LENGTH or 3 bytes for ITC_BARCODE_FIXED_LENGTH.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

None.

IS9CConfig::SetI2of5

This function updates the Interleaved 2 of 5 settings with new values.

Syntax

```
HRESULT IS9CConfig::SetI2of5( ITC_INTERLEAVED2OF5_DECODING
eDecode, ITC_INTERLEAVED2OF5_CHECK_DIGIT eCheck,
ITC_BARCODE_LENGTH_ID eLengthId, BYTE rgbLengthBuff[], DWORD
dwNumBytes );
```

Parameters

eDecode [in] Identifies the decoding for Interleaved 2 of 5 symbology.

eCheck [in] Identifies the check digit.

eLengthId [in] Use ITC_BARCODE_LENGTH_NO_CHANGE to indicate no change for bar code length. Use ITC_BARCODE_LENGTH for any length and minimum length, and set *rgbLengthBuff[0]* to a valid length value. Use ITC_BARCODE_FIXED_LENGTH to compose 1 or 2 or 3 fixed lengths, and set 3 bytes: *rgbLengthBuff[0]*, *rgbLengthBuff[1]*, *rgbLengthBuff[2]* with valid values.

rgbLengthBuff [in,size_is(dwNumBytes)] Contains bar code lengths when *eLengthId* = Use ITC_BARCODE_LENGTH or Use ITC_BARCODE_FIXED_LENGTH.

dwNumBytes [in] Number of bytes in *rbgLengthBuff[]*. For S9C, this value is 1 when *eLengthId* = ITC_BARCODE_LENGTH or 3 when *eLengthId* = ITC_BARCODE_FIXED_LENGTH.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

None.

Interleaved 2 of 5 Default Settings

Parameter	Default	Valid Range
Decoding	Not Active	ITC_INTERLEAVED2OF5_DECODING
Check Digit	Not Used	ITC_INTERLEAVED2OF5_CHECK_DIGIT
Bar Code Length	Minimum Length = 6	0x00`0xFE ITC_BC_LENGTH_NO_CHANGE

Interleaved 2 of 5 Enumerations

```
typedef enum tagInterleaved2of5Decoding
{
ITC_INTERLEAVED2OF5_NOTACTIVE = 0,       // Default
ITC_INTERLEAVED2OF5_ACTIVE = 1,
ITC_INTERLEAVED2OF5_NO_CHANGE = 255
} ITC_INTERLEAVED2OF5_DECODING;
typedef enum tagInterleaved2of5CheckDigit
{
ITC_INTERLEAVED2OF5_CHECK_NOTUSED,       // Default
ITC_INTERLEAVED2OF5_CHECK_MOD10_XMIT,
ITC_INTERLEAVED2OF5_CHECK_MOD10_NOTXMIT,
ITC_INTERLEAVED2OF5_CHECK_FRENCH_CIP_XMIT,
ITC_INTERLEAVED2OF5_CHECK_FRENCH_CIP_NOTXMIT,
ITC_INTERLEAVED2OF5_CHECK_NO_CHANGE = 255
} ITC_INTERLEAVED2OF5_CHECK_DIGIT;
typedef enum tagBarcodeLengthId
{
ITC_BARCODE_LENGTH = 0,
ITC_BARCODE_FIXED_LENGTH,
ITC_BARCODE_LENGTH_NO_CHANGE = 255
} ITC_BARCODE_LENGTH_ID;
```

IS9CConfig::GetMatrix2of5

This function retrieves the current settings of Matrix 2 of 5.

Syntax

```
HRESULT IS9CConfig::GetMatrix2of5( ITC_MATRIX2OF5_DECODING*
peDecode, DWORD* pdwLength );
```

Parameters

peDecode	[out]	Pointer to the ITC_MATRIX2OF5_DECODING location to receive the decoding for Matrix 2 of 5 symbology.
pdwLength	[out]	Pointer to the DWORD location to receive a value for the bar code length.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

None.

IS9CConfig::SetMatrix2of5

This function updates the Matrix 2 of 5 settings with new values.

Syntax

```
HRESULT IS9CConfig::SetMatrix2of5( ITC_MATRIX2OF5_DECODING
eDecode, DWORD dwLength );
```

Parameters

eDecode [in] Identifies the decoding for Matrix 2 of 5 symbology.

dwLength [in] Identifies the bar code length.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

None.

Matrix 2 of 5 Default Settings

Parameter	Default	Valid Range
Decoding	Not Active	ITC_MATRIX2OF5_DECODING
Bar Code Length	Minimum Length = 6	0x00`0xFE ITC_BC_LENGTH_NO_CHANGE

Matrix 2 of 5 Enumerations

```
typedef enum tagMatrix2of5Decoding
{
ITC_MATRIX2OF5_NOTACTIVE = 0, // Default
ITC_MATRIX2OF5_ACTIVE = 1,
ITC_MATRIX2OF5_NO_CHANGE = 255
} ITC_MATRIX2OF5_DECODING;
#define ITC_BC_LENGTH_NO_CHANGE 255. This definition can be used when the bar
code length does not require any change.
```

IS9CConfig::GetMSI

This function retrieves the current MSI settings.

Syntax

```
HRESULT IS9CConfig::GetMSI( ITC_MSI_DECODING* peDecode,
ITC_MSI_CHECK_DIGIT* peCheck, DWORD* pdwLength );
```

Parameters

peDecode	[out]	Pointer to the ITC_MSI_DECODING location to receive the decoding for MSI symbology.
peCheck	[out]	Pointer to the ITC_MSI_CHECK_DIGIT location to receive the check digit.
pdwLength	[out]	Pointer to the DWORD location to receive the bar code length.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

None.

IS9CConfig::SetMSI

This function updates the MSI settings with new values.

Syntax

```
HRESULT IS9CConfig::SetMSI( ITC_MSI_DECODING eDecode,
ITC_MSI_CHECK_DIGIT eCheck, DWORD dwLength );
```

Parameters

eDecode	[in]	Identifies the decoding for MSI symbology.
eCheck	[in]	Identifies the check digit.
dwLength	[in]	Identifies the bar code length.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

None.

MSI Default Settings

Parameter	Default	Valid Range
Decoding	Not Active	ITC_MSI_DECODING
Check Digit	MOD 10 checked and transmitted	ITC_MSI_CHECK_DIGIT
Bar Code Length	Minimum Length = 6	0x00`0xFE ITC_BC_LENGTH_NO_CHANGE

MSI Enumerations

```
typedef enum tagMsiDecoding
{
ITC_MSI_NOTACTIVE = 0, // Default
ITC_MSI_ACTIVE = 1,
ITC_MSI_NO_CHANGE = 255
} ITC_MSI_DECODING;
typedef enum tagMsiCheckDigit
{
ITC_MSI_CHECK_MOD10_XMIT, // Default
ITC_MSI_CHECK_MOD10_NOTXMIT,
ITC_MSI_CHECK_DOUBLEMOD10_XMIT,
ITC_MSI_CHECK_DOUBLEMOD10_NOTXMIT,
ITC_MSI_CHECK_NO_CHANGE = 255
} ITC_MSI_CHECK_DIGIT;
#define ITC_BC_LENGTH_NO_CHANGE 255. This definition can be used when the bar
code length does not require any change.
```

IS9CConfig::GetPDF417

This function retrieves the current PDF417 settings.

Syntax

```
HRESULT IS9CConfig::GetPDF417( ITC_PDF417_DECODING*
pePdf417Decode, ITC_PDF417_MACRO_PDF* peMacroPdf,
ITC_PDF417_CTRL_HEADER* pePdfControlHeader,
ITC_PDF417_FILE_NAME* pePdfFileName,
ITC_PDF417_SEGMENT_COUNT* pePdfSegmentCount,
ITC_PDF417_TIME_STAMP* pePdfTimeStamp, ITC_PDF417_SENDER*
pePdfSender, ITC_PDF417_ADDRESSEE* pePdfAddressee,
ITC_PDF417_FILE_SIZE* pePdfFileSize, ITC_PDF417_CHECKSUM*
pePdfChecksum );
```

Parameters

pePdf417Decode	[out]	Pointer to the ITC_PDF417_DECODING location to receive the decoding for PDF417 symbology.
peMacroPdf	[out]	Pointer to the ITC_PDF417_MACRO_PDF location to receive the Macro PDF.
pePdfControlHeader	[out]	Pointer to the ITC_PDF417_CTRL_HEADER location to receive the control header.
pePdfFileName	[out]	Pointer to the ITC_PDF417_FILE_NAME location to receive the file name.
pePdfSegmentCount	[out]	Pointer to the ITC_PDF417_SEGMENT_COUNT location to receive the segment count.
pePdfTimeStamp	[out]	Pointer to the ITC_PDF417_TIME_STAMP location to receive the time stamp.

pePdfSender	[out]	Pointer to the ITC_PDF417_SENDER location to receive the sender.
pePdfAddressee	[out]	Pointer to the ITC_PDF417_ADDRESSEE location to receive the addressee.
pePdfFileSize	[out]	Pointer to the ITC_PDF417_FILE_SIZE location to receive the file size.
pePdfChecksum	[out]	Pointer to the ITC_PDF417_CHECKSUM location to receive the checksum.

Return Values
HRESULT that indicates success or failure.

Remarks
None.

See Also
None.

IS9CConfig::SetPDF417
This function updates the PDF417 settings with new values.

Syntax
```
HRESULT IS9CConfig::SetPDF417( ITC_PDF417_DECODING
ePdf417Decode, ITC_PDF417_MACRO_PDF eMacroPdf,
ITC_PDF417_CTRL_HEADER ePdfControlHeader,
ITC_PDF417_FILE_NAME ePdfFileName, ITC_PDF417_SEGMENT_COUNT
ePdfSegmentCount, ITC_PDF417_TIME_STAMP ePdfTimeStamp,
ITC_PDF417_SENDER ePdfSender, ITC_PDF417_ADDRESSEE
ePdfAddressee, ITC_PDF417_FILE_SIZE ePdfFileSize,
ITC_PDF417_CHECKSUM ePdfChecksum );
```

Parameters
ePdf417Decode	[in]	Identifies the decoding for PDF417 symbology.
eMacroPdf	[in]	Identifies the Macro PDF.
ePdfControlHeader	[in]	Identifies the control header.
ePdfFileName	[in]	Identifies the file name.
ePdfSegmentCount	[in]	Identifies the segment count.
ePdfTimeStamp	[in]	Identifies the time stamp.
ePdfSender	[in]	Identifies the sender.
ePdfAddressee	[in]	Identifies the addressee.
ePdfFileSize	[in]	Identifies the file size.
ePdfChecksum	[in]	Identifies the checksum.

Return Values
HRESULT that indicates success or failure.

Remarks
None.

See Also
None.

PDF 417 Default Settings

Parameter	Default	Valid Range
Decoding	Not Active	ITC_PDF417_DECODING
Macro PDF	Macro PDF Buffered	ITC_PDF417_MACRO_PDF
Control Header	Not Transmitted	ITC_PDF417_CTRL_HEADER
*File Name	Not Transmitted	ITC_PDF417_FILE_NAME
*Segment Count	Not Transmitted	ITC_PDF417_SEGMENT_COUNT
*Time Stamp	Not Transmitted	ITC_PDF417_TIME_STAMP
*Sender	Not Transmitted	ITC_PDF417_SENDER
*Address	Not Transmitted	ITC_PDF417_ADDRESSEE
*File Size	Not Transmitted	ITC_PDF417_FILE_SIZE
*Check Sum	Not Transmitted	ITC_PDF417_CHECKSUM

** These are Macro PDF Optional Fields.*

PDF 417 Enumerations

```
typedef enum tagPdf417Decoding
{
ITC_PDF417_NOTACTIVE = 0,
ITC_PDF417_ACTIVE = 1,              // Default
ITC_PDF417_NO_CHANGE = 255
} ITC_PDF417_DECODING;
typedef enum tagPdf417MacroPdf
{
ITC_PDF417_MACRO_UNBUFFERED = 0,
ITC_PDF417_MACRO_BUFFERED = 1,      // Default
ITC_PDF417_MACRO_NO_CHANGE = 255
} ITC_PDF417_MACRO_PDF;
typedef enum tagPdf417ControlHeader
{
ITC_PDF417_CTRL_HEADER_NOTXMIT = 0,     // Default
ITC_PDF417_CTRL_HEADER_XMIT = 1,
ITC_PDF417_CTRL_HEADER_NO_CHANGE = 255
} ITC_PDF417_CTRL_HEADER;
typedef enum tagPdf417FileName
{
ITC_PDF417_FILE_NAME_NOTXMIT = 0,    // Default
ITC_PDF417_FILE_NAME_XMIT = 1,
ITC_PDF417_FILE_NAME_NO_CHANGE = 255
} ITC_PDF417_FILE_NAME;
typedef enum tagPdf417SegmentCount
{
ITC_PDF417_SEGMENT_COUNT_NOTXMIT = 0,// Default
ITC_PDF417_SEGMENT_COUNT_XMIT = 1,
```

```
ITC_PDF417_SEGMENT_COUNT_NO_CHANGE = 255
} ITC_PDF417_SEGMENT_COUNT;

typedef enum tagPdf417TimeStamp
{
ITC_PDF417_TIME_STAMP_NOTXMIT = 0,    // Default
ITC_PDF417_TIME_STAMP_XMIT = 1,
ITC_PDF417_TIME_STAMP_NO_CHANGE = 255
} ITC_PDF417_TIME_STAMP;
typedef enum tagPdf417Sender
{
ITC_PDF417_SENDER_NOTXMIT = 0,        // Default
ITC_PDF417_SENDER_XMIT = 1,
ITC_PDF417_SENDER_NO_CHANGE = 255
} ITC_PDF417_SENDER;
typedef enum tagPdf417Addressee
{
ITC_PDF417_ADDRESSEE_NOTXMIT = 0,     // Default
ITC_PDF417_ADDRESSEE_XMIT = 1,
ITC_PDF417_ADDRESSEE_NO_CHANGE = 255
} ITC_PDF417_ADDRESSEE;
typedef enum tagPdf417FileSize
{
ITC_PDF417_FILE_SIZE_NOTXMIT = 0,     // Default
ITC_PDF417_FILE_SIZE_XMIT = 1,
ITC_PDF417_FILE_SIZE_NO_CHANGE = 255
} ITC_PDF417_FILE_SIZE;
typedef enum tagPdf417Checksum
{
ITC_PDF417_CHECKSUM_NOTXMIT = 0,         // Default
ITC_PDF417_CHECKSUM_XMIT = 1,
ITC_PDF417_CHECKSUM_NO_CHANGE = 255
} ITC_PDF417_CHECKSUM;
```

IS9CConfig::GetPlessey

This function retrieves the current Plessey settings.

Syntax

```
HRESULT IS9CConfig::GetPlessey( ITC_PLESSEY_DECODING*
peDecode, ITC_PLESSEY_CHECK_DIGIT* peCheck, DWORD* pdwLength
);
```

Parameters

peDecode	[out]	Pointer to the ITC_PLESSEY_DECODING location to receive the decoding for Plessey symbology.
peCheck	[out]	Pointer to the ITC_PLESSEY_CHECK_DIGIT location to receive the check digit.
pdwLength	[out]	Pointer to the DWORD location to receive the bar code length.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

None.

IS9CConfig::SetPlessey

This function updates the Plessey settings with new values.

Syntax

```
HRESULT IS9CConfig::SetPlessey( ITC_PLESSEY_DECODING
eDecode, ITC_PLESSEY_CHECK_DIGIT eCheck, DWORD dwLength );
```

Parameters

eDecode	[in]	Identifies the decoding for Plessey symbology.
eCheck	[in]	Identifies the check digit.
dwLength	[in]	Identifies the bar code length.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

None.

Plessey Default Settings

Parameter	Default	Valid Range
Decoding	Not Active	ITC_PLESSEY_DECODING
Check Digit	Not Transmitted	ITC_PLESSEY_CHECK_DIGIT
Bar Code Length	Any Bar Code Length	0x00`0xFE ITC_BC_LENGTH_NO_CHANGE

Plessey Enumerations

```
typedef enum tagPlesseyDecoding
{
ITC_PLESSEY_NOTACTIVE = 0,           // Default
ITC_PLESSEY_ACTIVE = 1,
ITC_PLESSEY_NO_CHANGE = 255
} ITC_PLESSEY_DECODING;
typedef enum tagPlesseyCheckDigit
{
ITC_PLESSEY_CHECK_NOTXMIT = 0,    // Default
ITC_PLESSEY_CHECK_XMIT = 1,
ITC_PLESSEY_CHECK_NO_CHANGE = 255
} ITC_PLESSEY_CHECK_DIGIT;
#define ITC_BC_LENGTH_NO_CHANGE 255. This definition can be used when the bar
code length does not require any change.
```

IS9CConfig::GetStandard2of5

This function retrieves the current Standard 2 of 5 settings.

Syntax

```
HRESULT IS9CConfig::GetStandard2of5(
ITC_STANDARD2OF5_DECODING* peDecode,
ITC_STANDARD2OF5_FORMAT* peFormat,
ITC_STANDARD2OF5_CHECK_DIGIT* peCheck,
ITC_BARCODE_LENGTH_ID* peLengthId, BYTE rgbLengthBuff,
DWORD* pdwNumBytes );
```

Parameters

peDecode	[out]	Pointer to the ITC_STANDARD2OF5_DECODING location to receive the decoding for Standard 2 of 5 symbology.
peFormat	[out]	Pointer to the ITC_STANDARD2OF5_FORMAT location to receive the format.
peCheck	[out]	Pointer to the ITC_STANDARD2OF5_CHECK_DIGIT location to receive Modulo 10 check digit.
peLengthId	[out]	Pointer to the ITC_BARCODE_LENGTH_ID location to receive an indicator of either ITC_BARCODE_LENGTH or ITC_BARCODE_FIXED_LENGTH.
rgbLengthBuff	[out,size_is(3)]	An array of bytes to receives 1 byte of data for ITC_BARCODE_LENGTH, or 3 bytes of data for ITC_BARCODE_FIXED_LENGTH.
pdwNumBytes	[out]	Pointer to the DWORD location to receive a number indicating number of bytes in *rbgLengthBuff[]*: 1 byte for ITC_BARCODE_LENGTH or 3 bytes for ITC_BARCODE_FIXED_LENGTH.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

None.

IS9CConfig::SetStandard2of5

This function updates the Standard 2 of 5 settings with new values.

Syntax

```
HRESULT IS9CConfig::SetStandard2of5(
ITC_STANDARD2OF5_DECODING eDecode, ITC_STANDARD2OF5_FORMAT
eFormat, ITC_STANDARD2OF5_CHECK_DIGIT eCheck,
ITC_BARCODE_LENGTH_ID eLengthId, BYTE rgbLengthBuff[], DWORD
dwNumBytes );
```

Parameters

eDecode	[in]	Identifies the decoding for Standard 2 of 5 symbology.
eFormat	[in]	Identifies the format.
eCheck	[in]	Identifies the Modulo 10 check digit.
eLengthId	[in]	Use ITC_BARCODE_LENGTH_NO_CHANGE to indicate no change for bar code length. Use ITC_BARCODE_LENGTH for any length and minimum length, and set rgbLengthBuff[0] to a valid length value. Use ITC_BARCODE_FIXED_LENGTH to compose 1 or 2 or 3 fixed lengths, and set 3 bytes: *rgbLengthBuff[0]*, *rgbLengthBuff[1]*, *rgbLengthBuff[2]* with valid values.
rgbLengthBuff	[in,size_is(dwNumBytes)]	An array of bytes containing bar code lengths when *eLengthId* = ITC_BARCODE_LENGTH or ITC_BARCODE_FIXED_LENGTH.
dwNumBytes	[in]	Number of bytes in *rbgLengthBuff[]*. For S9C, this value is 1 when *eLengthId* = ITC_BARCODE_LENGTH or 3 when *eLengthId* = ITC_BARCODE_FIXED_LENGTH.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

None.

Standard 2 of 5 Default Settings

Parameter	Default	Valid Range
Decoding	Not Active	ITC_STANDARD2OF5_DECODING
Format	Identicon (6 Start/Stop bars)	ITC_STANDARD2OF5_FORMAT
Check Digit	Not Used	ITC_STANDARD2OF5_CHECK_DIGIT
Bar Code Length	Minimum Length = 6	0x00-0xFE ITC_BC_LENGTH_NO_CHANGE

Standard 2 of 5 Enumerations

```
typedef enum tagStandard2of5Decoding
{
ITC_STANDARD2OF5_NOTACTIVE = 0, // Default
ITC_STANDARD2OF5_ACTIVE = 1,
ITC_STANDARD2OF5_NO_CHANGE = 255
} ITC_STANDARD2OF5_DECODING;
typedef enum tagStandard2of5Format
{
ITC_STANDARD2OF5_FORMAT_IDENTICON, // Default
ITC_STANDARD2OF5_FORMAT_COMPUTER_IDENTICS,
ITC_STANDARD2OF5_FORMAT_NO_CHANGE = 255
} ITC_STANDARD2OF5_FORMAT;
typedef enum tagStandard2of5CheckDigit
{
ITC_STANDARD2OF5_CHECK_NOTUSED, // Default
ITC_STANDARD2OF5_CHECK_XMIT,
ITC_STANDARD2OF5_CHECK_NOTXMIT,
ITC_STANDARD2OF5_CHECK_NO_CHANGE = 255
} ITC_STANDARD2OF5_CHECK_DIGIT;
typedef enum tagBarcodeLengthId
{
ITC_BARCODE_LENGTH = 0,
ITC_BARCODE_FIXED_LENGTH,
ITC_BARCODE_LENGTH_NO_CHANGE = 255
} ITC_BARCODE_LENGTH_ID;
```

IS9CConfig::GetTelepen

This function retrieves the current Telepen settings.

Syntax

```
HRESULT IS9CConfig::GetTelepen( ITC_TELEPEN_DECODING*
peDecode, ITC_TELEPEN_FORMAT* peFormat );
```

Parameters

peDecode	[out]	Pointer to the ITC_TELEPEN_DECODING location to receive the decoding for TELEPEN symbology.
peFormat	[out]	Pointer to the ITC_TELEPEN_FORMAT location to receive the format.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

None.

IS9CConfig::SetTelepen

This function updates the Telepen settings with new values.

Syntax

```
HRESULT IS9CConfig::SetTelepen( ITC_TELEPEN_DECODING*
eDecode, ITC_TELEPEN_FORMAT* eFormat );
```

Parameters

eDecode	[in]	Identifies the decoding for Telepen symbology.
eFormat	[in]	Identifies the format.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

None.

Telepen Default Settings

Parameter	Default	Valid Range
Decoding	Not Active	ITC_TELEPEN_DECODING
Format	ASCII	ITC_TELEPEN_FORMAT

Telepen Enumerations

```
typedef enum tagTelepenDecoding
{
ITC_TELEPEN_NOTACTIVE = 0, // Default
ITC_TELEPEN_ACTIVE = 1,
ITC_TELEPEN_NO_CHANGE = 255
} ITC_TELEPEN_DECODING;
typedef enum tagTelepenDecoding
{
ITC_TELEPEN_FORMAT_ASCII, // Default
ITC_TELEPEN_FORMAT_NUMERIC,
ITC_TELEPEN_FORMAT_NO_CHANGE = 255
} ITC_TELEPEN_FORMAT;
```

IS9CConfig::GetUpcEan

This function retrieves the current UPC/EAN settings.

Syntax

```
HRESULT IS9CConfig::GetUpcEan( ITC_UPCEAN_DECODING*
upceanDecode, ITC_UPCA_SELECT* upcASelect, ITC_UPCE_SELECT*
upcESelect, ITC_EAN8_SELECT* ean8Select, ITC_EAN13_SELECT*
ean13Select, ITC_UPCEAN_ADDON_DIGITS* upcAddOnDigits,
ITC_UPCEAN_ADDON_TWO* upcAddOn2, ITC_UPCEAN_ADDON_FIVE*
upcAddOn5, ITC_UPCA_CHECK_DIGIT* upcACheck,
ITC_UPCE_CHECK_DIGIT* upcECheck, ITC_EAN8_CHECK_DIGIT*
ean8Check, ITC_EAN13_CHECK_DIGIT* ean13Check,
ITC_UPCA_NUMBER_SYSTEM* upcANumSystem,
ITC_UPCE_NUMBER_SYSTEM* upcENumSystem, ITC_UPCA_REENCODE*
upcAReencode, ITC_UPCE_REENCODE* upcEReencode,
ITC_EAN8_REENCODE* ean8Reencode );
```

Parameters

upceanDecode	[out]	Pointer to the ITC_UPCEAN_DECODING location to receive the decoding for UPC/EAN symbology.
upcASelect	[out]	Pointer to the ITC_UPCA_SELECT location to receive the UPC-A selection state.
upcESelect	[out]	Pointer to the ITC_UPCE_SELECT location to receive the UPC-E selection state.
ean8Select	[out]	Pointer to the ITC_EAN8_SELECT location to receive the EAN-8 selection state.
ean13Select	[out]	Pointer to the ITC_EAN13_SELECT location to receive the EAN-13 selection state.
upcAddOnDigits	[out]	Pointer to the ITC_UPCEAN_ADDON_DIGITS location to receive the add-on digits.
upcAddOn2	[out]	Pointer to the ITC_UPCEAN_ADDON_TWO location to receive the add-on 2 digits.
upcAddOn5	[out]	Pointer to the ITC_UPCEAN_ADDON_FIVE location to receive the add-on 5 digits.

upcACheck	[out]	Pointer to the ITC_UPCA_CHECK_DIGIT location to receive the UPC-A check digit.
upcECheck	[out]	Pointer to the ITC_UPCE_CHECK_DIGIT location to receive the UPC-E check digit.
ean8Check	[out]	Pointer to the ITC_EAN8_CHECK_DIGIT location to receive the EAN-8 check digit.
ean13Check	[out]	Pointer to the ITC_EAN13_CHECK_DIGIT location to receive the EAN-13 check digit.
upcANumSystem	[out]	Pointer to the ITC_UPCA_NUMBER_SYSTEM location to receive the UPC-A number system.
upcENumSystem	[out]	Pointer to the ITC_UPCE_NUMBER_SYSTEM location to receive the UPC-E number system.
upcAReencode	[out]	Pointer to the ITC_UPCA_REENCODE location to receive the UPC-A reencoding.
upcEReencode	[out]	Pointer to the ITC_UPCE_REENCODE location to receive the UPC-E reencoding.
ean8Reencode	[out]	Pointer to the ITC_EAN8_REENCODE location to receive the EAN-8 reencoding.

Return Values
HRESULT that indicates success or failure.

Remarks
None.

See Also
None.

IS9CConfig::SetUpcEan

This function updates the UPC/EAN settings with new values.

Syntax

```
HRESULT IS9CConfig::SetUpcEan( ITC_UPCEAN_DECODING
upceanDecode, ITC_UPCA_SELECT upcASelect, ITC_UPCE_SELECT
upcESelect, ITC_EAN8_SELECT ean8Select, ITC_EAN13_SELECT
ean13Select, ITC_UPCEAN_ADDON_DIGITS upcAddOnDigits,
ITC_UPCEAN_ADDON_TWO upcAddOn2, ITC_UPCEAN_ADDON_FIVE
upcAddOn5, ITC_UPCA_CHECK_DIGIT upcACheck,
ITC_UPCE_CHECK_DIGIT upcECheck, ITC_EAN8_CHECK_DIGIT
ean8Check, ITC_EAN13_CHECK_DIGIT ean13Check,
ITC_UPCA_NUMBER_SYSTEM upcANumSystem, ITC_UPCE_NUMBER_SYSTEM
upcENumSystem, ITC_UPCA_REENCODE upcAReencode,
ITC_UPCE_REENCODE upcEReencode, ITC_EAN8_REENCODE
ean8Reencode );
```

Parameters

upceanDecode	[in]	Identifies the decoding for UPC/EAN symbology.
upcASelect	[in]	Identifies the UPC-A selection state.
upcESelect	[in]	Identifies the UPC-E selection state.
ean8Select	[in]	Identifies the EAN-8 selection state.
ean13Select	[in]	Identifies the EAN-13 selection state.
upcAddOnDigits	[in]	Identifies the Add-on digits.
upcAddOn2	[in]	Identifies the Add-on 2 digits.
upcAddOn5	[in]	Identifies the Add-on 5 digits.
upcACheck	[in]	Identifies the UPC-A check digit.
upcECheck	[in]	Identifies the UPC-E check digit.
ean8Check	[in]	Identifies the EAN-8 check digit.
ean13Check	[in]	Identifies the EAN-13 check digit.
upcANumSystem	[in]	Identifies the UPC-A number system.
upcENumSystem	[in]	Identifies the UPC-E number system.
upcAReencode	[in]	Identifies the UPC-A reencoding.
upcEReencode	[in]	Identifies the UPC-E reencoding.
ean8Reencode	[in]	Identifies the EAN-8 reencoding.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

None.

UPC/EAN Default Settings

Parameter	Default	Valid Range
Decoding	ITC_UPCEAN_NO_CHANGE	This parameter is no longer used, set it to this value.
UPC-A	Active	ITC_UPCA_SELECT
UPC-E	Active	ITC_UPCE_SELECT
EAN-8	Active	ITC_EAN8_SELECT
EAN-13	Active	ITC_EAN13_SELECT
Add On Digits	Not Required	ITC_UPCEAN_ADDON_DIGITS
Add On 2 Digits	Not Active	ITC_UPCEAN_ADDON_TWO
Add On 5 Digits	Not Active	ITC_UPCEAN_ADDON_FIVE
UPC-A Check Digit	Transmitted	ITC_UPCA_CHECK_DIGIT
UPC-E Check Digit	Transmitted	ITC_UPCE_CHECK_DIGIT
EAN-8 Check Digit	Transmitted	ITC_EAN8_CHECK_DIGIT
EAN-13 Check Digit	Transmitted	ITC_EAN13_CHECK_DIGIT
UPC-A Number System	Transmitted	ITC_UPCA_NUMBER_SYSTEM
UPC-E Number System	Transmitted	ITC_UPCE_NUMBER_SYSTEM
Reencode UPC-A	UPC-A transmitted as EAN-13	ITC_UPCA_REENCODE
Reencode UPC-E	UPC-E transmitted as UPC-E	ITC_UPCE_REENCODE
Reencode EAN-8	EAN-8 transmitted as EAN-8	ITC_EAN8_REENCODE

UPC/EAN Enumerations

```
typedef enum tagUpcEanDecoding
{
ITC_UPCEAN_NOTACTIVE = 0,
ITC_UPCEAN_ACTIVE = 1,                // Default
ITC_UPCEAN_NO_CHANGE = 255
} ITC_UPCEAN_DECODING;
typedef enum tagUpcASelect
{
ITC_UPCA_DEACTIVATE,
ITC_UPCA_ACTIVATE,                    // Default
ITC_UPCA_NO_CHANGE = 255
} ITC_UPCA_SELECT;
typedef enum tagUpcESelect
{
ITC_UPCE_DEACTIVATE,
ITC_UPCE_ACTIVATE,                    // Default
ITC_UPCE_NO_CHANGE = 255
} ITC_UPCE_SELECT;
typedef enum tagEan8Select
{
ITC_EAN8_DEACTIVATE,
ITC_EAN8_ACTIVATE,                    // Default
ITC_EAN8_NO_CHANGE = 255
} ITC_EAN8_SELECT;
typedef enum tagEan13Select
{
ITC_EAN13_DEACTIVATE,
```

```
ITC_EAN13_ACTIVATE,                     // Default
ITC_EAN13_NO_CHANGE = 255
} ITC_EAN13_SELECT;
typedef enum tagUpcEanAddonDigits
{
ITC_UPCEAN_ADDON_NOT_REQUIRED,          // Default
ITC_UPCEAN_ADDON_REQUIRED,
ITC_UPCEAN_ADDON_NO_CHANGE = 255
} ITC_UPCEAN_ADDON_DIGITS;
typedef enum tagUpcEanAddonTwo
{
ITC_UPCEAN_ADDON_TWO_NOTACTIVE = 0,     // Default
ITC_UPCEAN_ADDON_TWO_ACTIVE = 1,
ITC_UPCEAN_ADDON_TWO_NO_CHANGE = 255
} ITC_UPCEAN_ADDON_TWO;
typedef enum tagUpcEanAddonFive
{
ITC_UPCEAN_ADDON_FIVE_NOTACTIVE = 0,    // Default
ITC_UPCEAN_ADDON_FIVE_ACTIVE = 1,
ITC_UPCEAN_ADDON_FIVE_NO_CHANGE = 255
} ITC_UPCEAN_ADDON_FIVE;
typedef enum tagUpcACheckDigit
{
ITC_UPCA_CHECK_NOTXMIT = 0,
ITC_UPCA_CHECK_XMIT = 1,                // Default
ITC_UPCA_CHECK_NO_CHANGE = 255
} ITC_UPCA_CHECK_DIGIT;
typedef enum tagUpcECheckDigit
{
ITC_UPCE_CHECK_NOTXMIT = 0,
ITC_UPCE_CHECK_XMIT = 1,                // Default
ITC_UPCE_CHECK_NO_CHANGE = 255
} ITC_UPCE_CHECK_DIGIT;
typedef enum tagEan8CheckDigit
{
ITC_EAN8_CHECK_NOTXMIT = 0,
ITC_EAN8_CHECK_XMIT = 1,                // Default
ITC_EAN8_CHECK_NO_CHANGE = 255
} ITC_EAN8_CHECK_DIGIT;
typedef enum tagEan13CheckDigit
{
ITC_EAN13_CHECK_NOTXMIT = 0,
ITC_EAN13_CHECK_XMIT = 1,               // Default
ITC_EAN13_CHECK_NO_CHANGE = 255
} ITC_EAN13_CHECK_DIGIT;
typedef enum tagUpcANumberSystem
{
ITC_UPCA_NUM_SYS_NOTXMIT = 0,
ITC_UPCA_NUM_SYS_XMIT = 1,              // Default
ITC_UPCA_NUM_SYS_NO_CHANGE = 255
} ITC_UPCA_NUMBER_SYSTEM;
typedef enum tagUpcENumberSystem
{
ITC_UPCE_NUM_SYS_NOTXMIT = 0,
ITC_UPCE_NUM_SYS_XMIT = 1,              // Default
ITC_UPCE_NUM_SYS_NO_CHANGE = 255
} ITC_UPCE_NUMBER_SYSTEM;
typedef enum tagUpcAReencode
{
```

```
ITC_UPCA_XMIT_AS_EAN13,                 // Default
ITC_UPCA_XMIT_AS_UPCA,
ITC_UPCA_XMIT_NO_CHANGE = 255
} ITC_UPCA_REENCODE;
typedef enum tagUpcEReencode
{
ITC_UPCE_XMIT_AS_UPCE,                  // Default
ITC_UPCE_XMIT_AS_UPCA,
ITC_UPCE_XMIT_NO_CHANGE = 255
} ITC_UPCE_REENCODE;
typedef enum tagEan8Reencode
{
ITC_EAN8_XMIT_AS_EAN8,                  //Default
ITC_EAN8_XMIT_AS_EAN13,
ITC_EAN8_XMIT_NO_CHANGE = 255
} ITC_EAN8_REENCODE;
```

IS9CConfig2 Functions

This interface is derived from the IS9CConfig interface and provides additional methods that can be used to set and retrieve the 700 Series Computer' s bar code configuration. All supported symbologies are initialized to their defaults when the S9C firmware is loaded.

GET/SET functions use enumerations as their parameters. In most enumerations, there is an enumerator xx_NO_CHANGE (such as ITC_CODE39_NO_CHANGE), where xx refers to a particular enumeration. This enumerator can be used during a call to a SET to indicate that no change is to be made to that particular parameter. This prevents the called function from having to format the same S9C command and send it down to the scanner.

To specify a bar code length of "any length," use a value of "0" for the bar code length argument.

IS9CConfig2 functions are the following. IS9CCONFIG.H is the header file and ITCUUID.LIB contains the IID_IADC Interface GUID value used to obtain the interface.

- IS9CConfig2::GetCode11 *(page 205)*
- IS9CConfig2::SetCode11 *(page 205)*
- IS9CConfig2::GetCustomSymIds *(page 207)*
- IS9CConfig2::SetCustomSymIds *(page 208)*
- IS9CConfig2::GetGlobalAmble *(page 211)*
- IS9CConfig2::SetGlobalAmble *(page 212)*
- IS9CConfig2::GetPDF417Ext *(page 213)*
- IS9CConfig2::SetPDF417Ext *(page 213)*
- IS9CConfig2::GetSymIdXmit *(page 214)*
- IS9CConfig2::SetSymIdXmit *(page 214)*

IS9CConfig2::GetCode11

This function retrieves the current settings for Code 11.

Syntax

```
HRESULT GetCode11( ITC_CODE11_DECODING* peDecode,
ITC_CODE11_CHECK_DIGIT* peCheck,
ITC_CODE11_CHECK_VERIFICATION* peVer );
```

Parameters

peDecode	[out]	Pointer to ITC_CODE11_DECODING location to receive Code 11 decoding.
peCheck	[out]	Pointer to ITC_CODE11_CHECK_DIGIT location to receive the check digit option.
peVer	[out]	Pointer to ITC_CODE11_CHECK_VERIFICATION location to receive the check verification option.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

None.

IS9CConfig2::SetCode11

This function updates the current setting of Code 11 symbology.

Syntax

```
HRESULT SetCode11( ITC_CODE11_DECODING eDecode,
ITC_CODE11_CHECK_DIGIT eCheck, ITC_CODE11_CHECK_VERIFICATION
eVer );
```

Parameters

eDecode	[in]	An enumeration that identifies decoding option for Code 11.
eCheck	[in]	An enumeration that identifies the check digit option.
eVer	[in]	An enumeration that identifies check verification option.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

None.

Code 11 Default Settings

Parameter	Default	Valid Range
Decoding	Not Active	ITC_CODE11_DECODING
Check Verification	1 Digit	ITC_CODE11_CHECK_VERIFICATION
Check Digit	Enable	ITC_CODE11_CHECK_DIGIT

Code 11 Enumerations

```
typedef enum tagCode11Decoding
{
ITC_CODE11_NOTACTIVE = 0,
ITC_CODE11_ACTIVE = 1, // Default
ITC_CODE11_NO_CHANGE = 255
} ITC_CODE11_DECODING;
typedef enum tagCode11CheckVerification
{
ITC_CODE11_CHK_VERIFY_ONEDIGIT = 1,
ITC_CODE11_CHK_VERIFY_TWODIGIT = 2, // Default
ITC_CODE11_CHK_VERIFY_NO_CHANGE = 255
} ITC_CODE11_CHECK_VERIFICATION;
typedef enum tagCode11CheckDigit
{
ITC_CODE11_CHECK_NOTXMIT = 0, // Default
ITC_CODE11_CHECK_XMIT = 1,
ITC_CODE11_CHECK_NO_CHANGE = 255
} ITC_CODE11_CHECK_DIGIT;
```

IS9CConfig2::GetCustomSymIds

This function retrieves all the custom symbology identifiers defined for the currently supported symbologies. *This is not supported when using an imager on the 700 Series Computer.*

Syntax

```
HRESULT GetCustomSymIds( ITC_CUST_SYM_ID_PAIR*
pStructSymIdPair,DWORD dwMaxNumElement, DWORD* pdwNumElement
);
```

Parameters

pStructSymIdPair	[out]	Pointer to ITC_CUST_SYM_ID_PAIR location to receive the current defined symbology identifiers for the supported symbologies. The caller must preallocate this buffer with *dwMaxNumElement* elements.
dwMaxNumElement	[in]	Maximum number of elements allocated for the *pStructSymIdPair* buffer which should always be equal to the last defined enumeration constant + 1 of the enumeration ITC_CUSTOM_ID. In this case, it is ITC_CUSTOMID_LAST_ELEMENT.
pdwNumElement	[out]	Pointer to DWORD location to receive the actual number of elements returned in the *pStructSymIdPair* buffer, which should be the same as *dwMaxNumElement*.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

- Custom Identifier Assignments *(page 209)*
- Custom Identifier Example *(page 210)*
- Custom Identifier Default Settings *(page 210)*

IS9CConfig2::SetCustomSymIds

This function updates the symbology identifiers (any ASCII values) for the currently supported symbologies. *This is not supported when using an imager on the 700 Series Computer.*

Syntax

```
HRESULT SetCustomSymIds( ITC_CUST_SYM_ID_PAIR*
pStructSymIdPair, DWORD dwNumElement );
```

Parameters

pStructSymIdPair	[in]	Pointer to ITC_CUST_SYM_ID_PAIR location, containing the new symbology identifiers for any supported symbologies to update.
dwNumElement	[in]	Identifies the number of symbology identifiers to update in the *pStructSymIdPair* buffer.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

None.

Custom Identifier Assignments

Each custom identifier is a one byte ASCII value within the range from 0x00 to 0xff. The enumerations in the ITC_CUSTOM_ID enumerator can be used as symbology identifications in the GetCustomSymIds() and SetCustomSymIds() functions.

```
typedef enum tagCustomId
{
ITC_CUSTOMID_CODABAR = 0       Identifies the Codabar symbology
ITC_CUSTOMID_CODE39            Identifies the Code 39 symbology
ITC_CUSTOMID_CODE93            Identifies the Code 93 symbology
ITC_CUSTOMID_CODE128_EAN_128   Identifies the Code 128 symbology
ITC_CUSTOMID_EAN8              Identifies the EAN-8 symbology
ITC_CUSTOMID_EAN13             Identifies the EAN-13 symbology
ITC_CUSTOMID_I2OF5             Identifies the Interleaved 2 of 5 symbology
ITC_CUSTOMID_MATRIX2OF5        Identifies the Matrix 2 of 5 symbology
ITC_CUSTOMID_MSI               Identifies the MSI symbology
ITC_CUSTOMID_PDF417            Identifies the PDF 417 symbology
ITC_CUSTOMID_PLESSEY           Identifies the Plessey symbology
ITC_CUSTOMID_CODE2OF5          Identifies the Standard 2 of 5 symbology
ITC_CUSTOMID_TELEPEN           Identifies the Telepen symbology
ITC_CUSTOMID_UPCA              Identifies the UPC-A symbology
ITC_CUSTOMID_UPCE              Identifies the UPC-E symbology
ITC_CUSTOMID_CODE11            Identifies the Code 11 symbology
ITC_CUSTOMID_LAST_ELEMENT      Identifies the last element. Use to preallocate
the buffer on GetCustomSymIds
}ITC_CUSTOM_ID;
typedef struct tagCustSymbIdPair
{

ITC_CUSTOM_ID eSymbology;      Identifies the symbology of interest

BYTE byteId;
 ASCII value (1 byte within the range0x00 – 0xf)

}ITC_CUST_SYM_ID_PAIR;
```

Custom Identifier Default Settings

Symbology	Default	Valid Range
Codabar	D	0x00-0xFF
Code 11	*	0x00-0xFF
Code 39	*	0x00-0xFF
Code 93	D	0x00-0xFF
Code128/EAN 128	D	0x00-0xFF
EAN-8	0xFF	0x00-0xFF
EAN-13	F	0x00-0xFF
Interleaved 2 of 5	I	0x00-0xFF
Matrix 2 of 5	D	0x00-0xFF
MSI	D	0x00-0xFF
PDF 417	*	0x00-0xFF
Plessey	D	0x00-0xFF
Standard 2 of 5	D	0x00-0xFF
Telepen	*	0x00-0xFF
UPC-A	A	0x00-0xFF
UPC-E	E	0x00-0xFF

Custom Identifier Example

The following code segment is an example of updating the UPC-E and UPC-A symbology identifiers with new values, and then retrieving the currently defined symbology identifiers for all the supported symbologies:

```
ITC_CUST_SYM_ID_PAIR oStructSymIdPair [ITC_CUSTOMID_LAST_ELEMENT];
oStructSymIdPair[0].eSymbology = ITC_CUSTOMID_UPCE;
oStructSymIdPair[0].byteId = 0x41;            // ASCII char A
oStructSymIdPair[1].eSymbology = ITC_CUSTOMID_UPCA;
oStructSymIdPair[1].byteId = 0x42;            // ASCII char B
HRESULT hr = pIS9CConfig2->SetCustomSymIds(&oStructSymIdPair[0], 2];
DWORD dwNum = 0;
HRESULT hr = pIS9CConfig2->GetCustomSymIds(&oStructSymIdPair[0],
ITC_CUSTOMID_LAST_ELEMENT, &dwNum);
```

IS9CConfig2::GetGlobalAmble

This retrieves the scanner's current preamble or postamble setting.

Syntax

```
HRESULT GetGlobalAmble( ITC_GLOBAL_AMBLE_ID eAmbleId, BYTE
rgbBuffer[], DWORD dwBufferSize, DWORD* pdwBufferSize );
```

Parameters

eAmbleId	[in]	An enumeration of type ITC_GLOBAL_AMBLE_ID identifies whether the preamble or postamble setting is to be retrieved. Only one setting can be queried at a time.
rgbBuffer	[in]	Contains the buffer for the postamble or preamble setting to be queried.
dwBufferSize	[in]	The maximum number of bytes that rgbBuffer can store. Must be at least ITC_GLOBAL_AMBLE_MAX_CHARS bytes.
pdwBufferSize	[out]	A pointer to DWORD location to store the actual number of returned bytes in *rgbBuffer*.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

None.

IS9CConfig2::SetGlobalAmble

This function updates the scanner's current preamble or postamble setting depending on the input parameters.

Syntax

```
HRESULT SetGlobalAmble( ITC_GLOBAL_AMBLE_ID eAmbleId, BYTE
rgbBuffer[], DWORD dwBufferSize );
```

Parameters

eAmbleId	[in]	An enumeration of type ITC_GLOBAL_AMBLE_ID identifies whether the preamble or postamble setting is to be updated. Only one setting can be updated at a time.
rgbBuffer	[in]	Contains the buffer for the postamble or preamble setting to be updated.
dwBufferSize	[in]	Identifies number of bytes in *rgbBuffer*.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

None.

Postamble and Preamble Defaults

Parameter	Default	Valid Range
Preamble	Null	0 to 20 ASCII characters
Postamble	Null	0 to 20 ASCII characters

IS9CConfig2::GetPDF417Ext

This function is an extended function for retrieving the PDF 417 settings not included in the IS9CConfig::GetPDF417.

Syntax

```
HRESULT GetPDF417Ext( ITC_MICRO_PDF417_DECODING* peDecode,
ITC_MICRO_PDF417_CODE128_EMULATION* peCode128 );
```

Parameters

peDecode [out] Pointer to ITC_MICRO_PDF417_DECODING location to receive the Micro PDF 417 decoding.

peCode128 [out] Pointer to ITC_MICRO_PDF417_CODE128_EMULATION* location to receive the Micro PDF 417 Code 128 emulation option.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

None.

IS9CConfig2::SetPDF417Ext

This function is an extended function for updating the additional PDF 417 settings not included in IS9CConfig::SetPDF417.

Syntax

```
HRESULT SetPDF417Ext( ITC_MICRO_PDF417_DECODING eDecode,
ITC_MICRO_PDF417_CODE128_EMULATION eCode128 );
```

Parameters

eDecode [in] An enumeration that identifies decoding option for the Micro PDF 417.

eCode128 [in] An enumeration that identifies the Code 128 emulation option for the Micro PDF 417.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

None.

PDF 417 Extended: Micro PDF 417 Default Settings

Parameter	Default	Valid Range
Decoding	Not Active	ITC_MICRO_PDF417_DECODING
Code 128 Emulation	Not Active	ITC_MICRO_PDF417_CODE128_EMULATION
** These are Micro PDF 417 parameters.*		

IS9CConfig2::GetSymIdXmit

This function retrieves the current symbology ID transmission option as described on the next page.

Syntax

```
HRESULT GetSymIdXmit( ITC_SYMBOLOGY_ID_XMIT* peSymIdXmit );
```

Parameters

peSymIdXmit [out] Pointer to ITC_SYMBOLOGY_ID_XMIT location to receive the current symbology identifier transmission option.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

None.

IS9CConfig2::SetSymIdXmit

This updates the symbology ID transmission option shown next page.

Syntax

```
HRESULT SetSymIdXmit( ITC_SYMBOLOGY_ID_XMIT eSymIdXmit );
```

Parameters

eSymIdXmit [in] Identifies the symbology identifier transmission option to update.

Return Values

HRESULT that indicates success or failure.

Remarks

None.

See Also

None.

Symbology ID Transmission Option

The symbology identifier (or code mark) concept provides a standardized way for a device receiving data from a bar code reader to differentiate between the symbologies.

The following symbology ID transmission option specifies whether or not the symbology ID should be transmitted as part of the scanned bar code label to all the connected data collection applications. Options for transmission are: do not transmit, transmit the standard AIM identifiers, or transmit the one byte custom defined identifiers. AIM and custom identifiers cannot be selected to be transmitted at the same time; only the last selected option will be active.

```
typedef enum tagSymbologyIdXmit

{
```

`ITC_ID_XMIT_DISABLE = 0`	Symbology identifier will not be transmitted as part of the label. This is the default setting.
`ITC_ID_XMIT_CUSTOM = 1`	Activate custom symbology identifier transmission for all symbologies. Example of the transmitted label: [preamble] [Custom ID] <data> [postamble]
`ITC_ID_XMIT_AIM = 2`	Activate AIM symbology identifier transmission for all symbologies. Example of the transmitted label: [preamble] [AIM symbology ID] <data> [postamble]

```
}ITC_SYMBOLOGY_ID_XMIT;
```

IS9CConfig3 Functions

The IS9CConfig3 interface provides generic methods for retrieving and setting configuration using ISCP commands.

ISCP Commands

An ISCP Command is composed of three or more bytes formatted as <SG><FID><parameters> where:

- *SG* Setup group.
- *FID* Function ID.
- *parameters* One or more configuration value bytes depending on the configuration.

ISCP commands include the following:

Imager Settings

This dictates the start and end column positions for the image dimension.

SG	FID	Parameter	Description
0x7B	80	Value [0..639]	Start column position.
0x7B	81	Value [0..639]	End column position.

Trigger Settings

This sets the duration of the aiming beam before acquiring images to be decoded.

SG	FID	Parameter	Description
0x70	81	Value [0..65535]	Number of milliseconds.

QRCode Symbology

This enables or disables the QRCode symbology.

SG	FID	Parameter	Description
0x55	40	0	Disable this symbology.
0x55	40	1	Enable this symbology.

Data Matrix Symbology

This enables or disables the Data Matrix symbology.

SG	FID	Parameter	Description
0x54	40	0	Disable this symbology.
0x54	40	1	Enable this symbology.

ISCP::GetConfig

This retrieves configurations using the ISCP commands format.

Syntax

```
HRESULT ISCPGetConfig( BYTE rgbCommandBuff[], DWORD
dwCommandBuffSize, BYTE rgbReplyBuff[], DWORD
dwReplyBuffMaxSize, DWORD *pdwReplyBuffSize );
```

Parameters

rgbCommandBuff	[in, size_is]	Contains ISCP commands in array of bytes.
dwCommandBuffSize	[in]	Number of bytes in *rgbCommandBuff*.
rgbReplyBuff	[in, out, size_is]	Results of query in array of bytes.
dwReplyBuffMaxSize	[in]	Maximum size of *rgdReplyBuff*.
pdwReplyBuffSize	[in, out]	Number of bytes placed in *rbfReplyBuff*.

Return Values

None.

Remarks

None.

See Also

None.

ISCP::SetConfig

This updates configurations using the ISCP commands format.

Syntax

```
HRESULT ISCPSetConfig( BYTE rgbCommandBuff[], DWORD
dwCommandBuffSize, BYTE rgbReplyBuff[], DWORD
dwReplyBuffMaxSize, DWORD *pdwReplyBuffSize );
```

Parameters

rgbCommandBuff	[in, size_is]	Contains ISCP commands in array of bytes.
dwCommandBuffSize	[in]	Number of bytes in *rgbCommandBuff*.
rgbReplyBuff	[in, out, size_is]	Results of request in array of bytes.
dwReplyBuffMaxSize	[in]	Maximum size of *rgbReplyBuff*.
pdwReplyBuffSize	[in, out]	Number of bytes placed in *rgbReplyBuff*.

Return Values

None.

Remarks

None.

See Also

None.

AIM Symbology ID Defaults

Refer to the official AIM documentation on symbology identifiers for full information on the different processing options supported.

Symbology	ID Character	Modifier Characters	
Codabar	F	0	Standard Codabar symbol. No special processing.
		1	ABC Codabar (American Blood commission) concatenate/message append performed.
		2	Reader has validated the check character.
		4	Reader has stripped the check character before transmission.
Code 11	H	0	Single modulo 11 check character validated and transmitted.
		1	Two modulo 11 check characters validated and transmitted.
		3	Check characters validated but not transmitted.
Code 39	A	0	No check character validation nor full ASCII processing. All data transmitted as decoded.
		1	Modulo 43 check character validated and transmitted.
		3	Modulo 43 check character validated but not transmitted.
		4	Full ASCII character conversion performed. No check character validation.
		5	Full ASCII character conversion performed. Modulo 43 check character validated and transmitted.
		7	Full ASCII character conversion performed. Modulo 43 check character validated but not transmitted.
Code 93	G	0	No options specified. Always transmit 0.
Code128	C	0	Standard data packet. No FNC1 in first or second symbol character position after start character.
		1	EAN/UCC-128 data packet. FNC1 in first symbol character position after start character.
		2	FNC1 in second symbol character position after start character.
		4	Concatenation according to International Society for Blood Transfusion specifications was performed. Concatenated data follows.
Interleaved 2 of 5	I	0	No check character validation.
		1	Modulo 10 symbol check character validated and transmitted
		3	Modulo 10 symbol check character validated but not transmitted.
Matrix 2 of 5	X	0`F	For symbologies or symbology options not listed, a code character with the value 0-F may be assigned by the decoder manufacturer to identify those symbologies and options implemented in the reader.
MSI	M	0	Modulo 10 symbol check character validated and transmitted.
		1	Modulo 10 symbol check character validated but not transmitted.

Symbology (continued)	ID Character	Modifier Characters
PDF 417/ Micro PDF 417	L	0 Reader set to conform with protocol defined in 1994 PDF 417 specifications. 1 Reader set to follow protocol of ENV 12925 for Extended Channel Interpretation (all data characters 92 doubled). 2 Reader set to follow protocol of ENV 12925 for Basic Channel Interpretation (data characters 92 are not doubled). 3 Code 128 emulation: implied FNC1 in first position. 4 Code 128 emulation: implied FNC1 after initial letter or pair of digits. 5 Code 128 emulation: no implied FNC1.
Plessey	P	0 No options specified. Always transmit 0.
Standard 2 of 5 (2-bar start/stop)	R	0 No check character validation. 1 Modulo 7 check character validated and transmitted. 3 Modulo 7 check character validated but not transmitted.
Standard 2 of 5 (3-bar start/stop)	S	0 No options specified. Always transmit 0.
Telepen	B	0 Full ASCII mode 1 Double density numeric only mode 2 Double density numeric followed by full ASCII 4 Full ASCII followed by double density numeric
UPC/EAN	E	Consider UPC/EAN symbols with supplements as two separate symbols. The first symbol is the main data packet, and the second symbol is the 2 or 5 digit supplement. Transmit these two symbols separately, each with its own symbology identifier. Provision is made for the option of transmitting both symbols as a single data packet. 0 Standard data packet in full EAN format (13 digits for EAN-13, UPC-A, and UPC-E; does not include add-on data). 1 Two digit add-on data only. 2 Five digit add-on data only. 3 Combined data packet comprising 13 digits from EAN-13, UPC-A, or UPC-E symbol and 2 or 5 digits from add-on symbol. 4 EAN-8 data packet
*IMPORTANT: The "symbology_id" character letter **must** be uppercase for the above definitions.*		

IImage Interface

The IImage interface gives the application the capability to acquire images. The image acquired can be either a raw image as captured by the digital camera or it can be normalized. A normalized image is presented the same as if the picture were taken at right angles to the image and at the same distance. The normalized image is commonly used for signature capture applications.

- IImage::ReadSigCapBuffer *(page 221)*
- IImage::ReadSigCapFile *(page 224)*
- IImage::ReadImage *(page 225)*
- IImage::CancelReadImage *(page 226)*
- IImage::Start *(page 226)*
- IImage::Stop *(page 227)*
- IImage::Open *(page 227)*
- IImage::Close *(page 228)*

IImage::ReadSigCapBuffer

Syntax

```
HRESULT IImage::ReadSigCapBuffer( ITC_SIGCAP_SPEC
*pSigCapSpec, ITC_IMAGE_SPEC *pImgBuffer, DWORD nMaxBuffSize
);
```

Parameters

Parameters:

pSigCapSpec [in] Pointer to the structure that identifies the signature capture region. This structure is defined as follows:

```
typedef struct tagITCSigCapSpec
{
DWORD dwStructSize;
INT iAspectRatio;
INT iOffsetX;
INT iOffsetY;
UINT uiWidth;
UINT uiHeight;
INT iResolution;
ITCFileFormat eFormat;
DWORD eDepth;
} ITC_SIGCAP_SPEC;
```

where:

- *dwStructSize* Size, in bytes, of this struct. This is for version control.
- *iAspectRatio* Ratio of the bar code height (linear bar codes) or row height (2D bar codes) to the narrow element width.
- *iOffsetX* Offset in X direction, relative to barcode center. Positive values are right of the bar code, negative values to the left.

- *iOffsetY* Offset in Y direction, relative to barcode center. Positive values are higher than the bar code, negative values lower.
- *uiWidth* Width of signature capture image region in intelligent bar code units.
- *uiHeight* Height of the signature capture image region in intelligent bar code units.
- *iResolution* Number of pixels per intelligent bar code unit.
- *eFormat* Format of the image buffer returned as follows. Currently, only ITC_FILE_RAW is supported.

```
ITC_FILE_KIM =                0,    // Returns data a KIM file
ITC_FILE_TIFF_BIN =           1,    // TIFF Binary file
ITC_FILE_TIFF_BIN_GROUP4 =    2,    // TIFF Binary Group 4 compressed
ITC_FILE_TIFF_GRAY_SCALE =    3,    // TIFF Gray Scale
ITC_FILE_RAW =                4,    // Raw image
ITC_FILE_JPEG =               5,    // JPEG image
```

- *eDepth* Number of bits per pixel. Currently, only one (monochrome) or eight (gray-scale) are supported.

pImgBuffer [out] Pointer to the buffer in which the signature capture image will be put.

```
typedef struct tagITCImageSpec
{
   DWORD dwStructSize;
   LONG  biWidth;
   LONG  biHeight;
   WORD  biBitCount;
   ITC_FILE_FORMAT eFormat;
   DWORD biActualImageSize;
   DWORD biMaxImageBytes;
   BYTE  rgbImageData[1];
} ITC_IMAGE_SPEC;
```

where:

- *dwStructSize* Size, in bytes, of this struct. This is for version control.
- *biWidth* The width of each row in pixels.
- *biHeight* The number of rows in the image data.
- *biBitCount* The number of bits per pixel.
- *eFormat* Identifies the image format.
- *biActualImageSize* Total bytes of image data returned.
- *biMaxImageBytes* Maximum bytes that can be stored in *rgbImageData[]*.
- *rgbImageData* Buffer containing the actual data, for example a 640x480 uses a 307200-byte buffer. The array size of this buffer is arbitrary so do *not* use this structure directly to reserve memory. The actual dimension of the buffer is identified by *biMaxImageBytes*.

Return Values

HRESULT identifying success or error. On error, the following codes will be returned:

- **S_OK**
  Image successfully returned.
- **ITC_RESULT_ERR_BADREGION_E**
  The specified region is not in the image.
- **ITC_RESULT_NO_BC_DECODED_E**
  A bar code has not yet been decoded or the last bar code decoded was not a signature capture symbology.
- **ITC_IMGBUFF_TOO_SMALL_E**
  pImgBuffer is too small to contain the signature captured image.
- **ITC_INV_PARAMETER_E**
  One of the parameters is invalid.
- **S_DEVICE_NOT_OPENED_E**
  The device had not been opened.

Remarks

ReadSigCapBuffer() will return the image from the last decoded label with dimensions identified by the calling parameter. This signature capture region must include the signature capture bar code. The supported bar codes for signature capture are: PDF 417, Code 128, and Code 39. The caller specifies the width, height, and center of the image to be retrieved. This image is independent of any rotation of the bar code relative to the imager. Thus, if the bar code is decoded with the code itself upside down to the imager, the retrieved image will still be right side up. However, if the specified image is outside the field of view a result code of ITC_RESULT_ERR_BADREGION_E will be returned.

This function uses the dimensions of the last decoded bar code as its coordinate system. Thus, all the parameters describing the image size and position are in units called “Intelligent Bar Code Units.” An Intelligent Bar Code Unit is equivalent to the narrow element width of the bar code.

The dimensions of the resulting image can be calculated with this formula:

```
Resulting Width = Specified Width * Specified Resolution
Resulting Height = Specified Height * Specified Resolution
```

See Also

None.

IImage::ReadSigCapFile

Note: This has not been implemented as of this publication.

Syntax

```
HRESULT IImage::ReadSigCapFile( ITC_SIGCAP_SPEC
*pSigCapSpec, LPCTSTR pszFileName );
```

Parameters

pSigCapSpec	[in]	Pointer to the structure that identifies the signature capture region. See ReadSigCapFile (page 221) for a description of this structure.
pszFileName	[in]	Name of the file in which to copy the image.

Return Values

HRESULT identifying success or error. On error, the following codes will be returned:

- **S_OK**
  Image successfully returned.
- **ITC_RESULT_ERR_BADREGION_E**
  The specified region is not in the image.
- **ITC_RESULT_NO_BC_DECODED_E**
  A bar code has not yet been decoded or the last bar code decoded was not a signature capture symbology.
- **ITC_FILE_OPEN_E**
  The file could not be opened.
- **ITC_INV_PARAMETER_E**
  One of the parameters is invalid.
- **S_DEVICE_NOT_OPENED_E**
  The device had not been opened.

Remarks

ReadSigCapFile() will write the image from the last decoded label with dimensions identified by the calling parameter. If the file already exists, its contents will be overwritten.

This signature capture region must include the signature capture bar code. The supported bar codes for signature capture are: PDF 417, Code 128, and Code 39. The caller specifies the width, height, and center of the image to be retrieved. This image is independent of any rotation of the bar code relative to the imager. Thus, if the bar code is decoded with the code itself upside down to the imager, the retrieved image will still be right side up. However, if the specified image is outside the field of view a result code of ITC_RESULT_ERR_BADREGION_E will be returned.

This function uses the dimensions of the last decoded bar code as its coordinate system. Thus, all the parameters describing the image size and position are in units called “Intelligent Bar Code Units”. An Intelligent Bar Code Unit is equivalent to the narrow element width of the bar code.

The dimensions of the resulting image can be calculated with this formula:

```
Resulting Width = Specified Width * Specified Resolution
Resulting Height = Specified Height * Specified Resolution
```

See Also
None.

IImage::ReadImage

Syntax
```
HRESULT IImage::Read( ITCFileFormat eFormat, DWORD nDepth,
ITC_IMAGE_SPEC *pImgBuffer, DWORD dwTimeout );
```

Parameters
eFormat [in] Format of the image buffer returned as follows. Currently, only ITC_FILE_RAW is supported.

```
ITC_FILE_KIM =              0,    // Returns data a KIM file
ITC_FILE_TIFF_BIN =         1,    // TIFF Binary file
ITC_FILE_TIFF_BIN_GROUP4 =  2,    // TIFF Binary Group 4 compressed
ITC_FILE_TIFF_GRAY_SCALE =  3,    // TIFF Gray Scale
ITC_FILE_RAW =              4,    // Raw image
ITC_FILE_JPEG =             5,    // JPEG image
```

nDepth [in] Number of bits per pixel. Currently, only eight (gray-scale) are supported.

pImgBuffer [in/out] Pointer to the buffer containing the image.

dwTimeout [in] Milliseconds to wait for the image to be returned.

Return Values
HRESULT identifying success or error. On error, these will be returned:

- **S_OK** Image successfully returned.
- **ITC_IMGBUFF_TOO_SMALL_E** *pImgBuffer* is too small to contain the signature captured image.
- **ITC_TIMEOUT_E** Timeout.
- **ITC_INV_PARAMETER_E** One of the parameters is invalid.
- **S_DEVICE_NOT_OPENED_E** The device had not been opened.

Remarks
The image is returned in *pImgBuffer* in the caller specified format.

See Also
None.

IImage::CancelReadImage

Syntax
```
HRESULT IImage::CancelReadImage( );
```

Parameters
None.

Return Values
Status code indicating success or failure as follows:

- **S_OK** Imager closed.
- **S_DEVICE_NOT_OPENED_E** The device had not been opened.

Remarks
This function causes a pending image read of IImage::ReadImage() to return immediately with an error status. The purpose of this function is to allow the application to release a thread blocked on the ReadImage() call.

See Also
None.

IImage::Start

Syntax
```
HRESULT IImage::Start( );
```

Parameters
None.

Return Values
Status code indicating success or failure as follows:

- **S_OK** Imager started.
- **S_DEVICE_NOT_OPENED_E** The device had not been opened.

Remarks
This function starts the image continuously capturing images.

See Also
None.

IImage::Stop

Syntax
```
HRESULT IImage::Stop( );
```

Parameters
None.

Return Values
Status code indicating success or failure as follows:

- **S_OK** Imager started.
- **S_IMG_NOT_PRESENT_E** Unit does not contain an imager.
- **S_DEVICE_NOT_OPENED_E** Device had not been opened.

Remarks
This function stops the image continuously capturing images.

See Also
None.

IImage::Open

Syntax
```
HRESULT IImage::Open( BOOL fSigCapEnable );
```

Parameters
fSigCapEnable [in] When TRUE, signature capture is enabled. When FALSE, it is disabled. Bar code labels are decoded and images (via IImage::ReadImage) the same.

Return Values
Status code indicating success or failure as follows:

- **S_OK** Imager opened.
- **S_IMG_NOT_PRESENT_E** Unit does not contain an imager.
- **S_DEVICE_CONTENTION_E** Device has already been opened.

Remarks
This function exclusively allocates the imager device so that the other IImage methods can be safely called.

See Also
None.

IImage::Close

Syntax

```
HRESULT IImage::Close( );
```

Parameters

None.

Return Values

Status code indicating success or failure as follows:

- **S_OK** Imager closed.
- **S_DEVICE_NOT_OPENED_E** The device had not been opened.

Remarks

This function releases the imager device so that other applications can open it. An IImage::Release() will also close the imager device.

See Also

None.

Data Collection Configuration

Scanner settings for the 700 Series Computer can be configured via the **Data Collection** control panel applet. From the 700 Series Computer, tap **Start** → **Settings** → the **System** tab → the **Data Collection** icon. See *Appendix A*, "*Control Panel Applets*" for more information about the following parameters. *Note that these are in alphabetical order.*

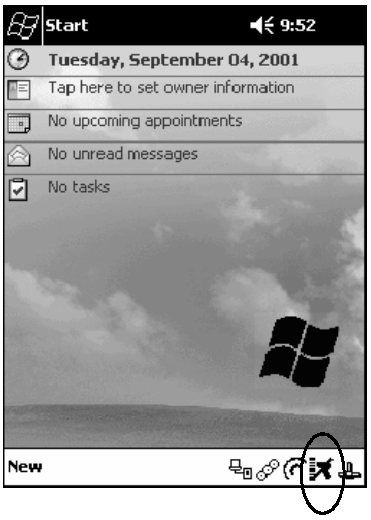
- Codabar *(page 292)*
- Code 11 *(page 306)*
- Code 128 *(page 295)*
  - Code 128 Options *(page 296)*
  - Code 128 FNC1 Character *(page 297)*
- Code 39 *(page 290)*
- Code 93 *(page 294)*
  - Code 93 Length *(page 294)*
- Data Matrix *(page 308)*
- Interleaved 2 of 5 *(page 303)*
- Matrix 2 of 5 *(page 304)*
- MSI *(page 299)*
- PDF 417 *(page 300)*
  - Macro PDF *(page 300)*
  - Micro PDF 417 *(page 302)*
- Plessey *(page 298)*
- QR Code *(page 307)*
- Standard 2 of 5 *(page 291)*
- Telepen *(page 305)*
- UPC/EAN *(page 293)*

Tethered Scanner

The Intermec Tethered Scanner feature accepts data from the COM1 port wedges it to the keyboard interface, and allows some ADC. This feature can be enabled or disabled from the Today Screen on the 700 Series Computer.

Enabling and Disabling

On the 700 Series Computer, tap **Start → Today**. Tap the bar code scanner icon in the System Tray *(circled in the following illustration)*. Initially, the bar code scanner icon indicates that this feature is disabled *(shown to the left)*.

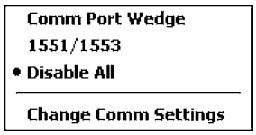


- Select **Comm Port Wedge** to send any data, coming into the 700 Series Computer through the COM1 port from an external input device, as keyboard data to an application on the desktop.

  For example, if you have Pocket Word running on your 700 Series Computer desktop, information scanned with a scanner connected to the COM1 port will appear in the Word document. If another data collection application is running and is active on the 700 Series Computer, the scanned information will appear in that application.

Note: When **Comm Port Wedge** is selected, regardless of the data sent by the external input device, you cannot control the device or the data format using any of the Intermec scanner control or data transfer APIs from the SDK or the internal Data Collection software. The external input device is governed by what software it has onboard to tell it how to scan, take pictures, or send the data elsewhere.

- Select **1551/1553** to enable the Sabre 1551E or 1553 Tethered Scanner to scan, then send data as keyboard data. The 1551/1553 Tethered Scanner has software onboard that translates scanned data into characters, so the running/active application does not need to know how to do that. All the scanner control and data transfer APIs will work with the 1551/1553 Tethered Scanner, so you can control the device.
- Select **Disable All** to disable this feature and use the COM1 port for another application, such as ActiveSync. An error message will result if this option were not selected, but this action was attempted. Similarly, if ActiveSync is using the COM1 port, and you select **Comm Port Wedge** or **1551/1553**, an error message will result. See "*Error Message*" on page 232 for more information.



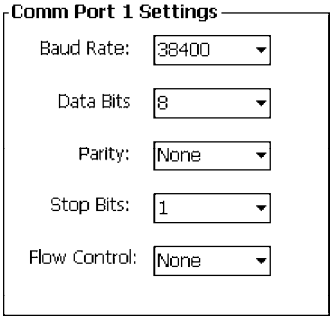
Changing Comm Settings

Tap **Change Comm Settings** to configure the settings for the COM1 port. Current settings are restored after a warm-boot, but are lost after a cold-boot. When these settings have not been changed, the **OK** button is disabled (grayed out). When changes are made, tap **OK** after it is enabled to accept these changes.

- **Baud Rate**: 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200
- **Data Bits**: 7 or 8
- **Parity**: None, Odd, Even, Mark, Space
- **Stop Bits**: 1 or 2
- **Flow Control**: None or Hardware

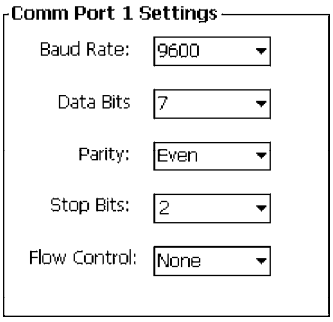
Tethered Scanner

The default settings for the Tethered Scanner are shown in the following illustration:



Sabre 1551E or 1553 Tethered Scanner

The default communication configuration for the Sabre 1551E or 1553 Tethered Scanner is shown in the following illustration. Scan the EasySet Reset Factory Defaults label to set the Sabre 1551E or 1553 tethered scanner communications settings to this configuration. The COM1 port configuration settings must also match those of the scanner to scan labels.



Welch Allyn 1470 Imager Settings

The Welch Allyn 1470 Imager can be set to this configuration by scanning the Factory Default Settings label.

Error Message

If the COM1 port is used by another application, such as ActiveSync, neither the Comm Port Wedge nor the 1551/1553 Tethered Scanner can be enabled. As a result, the following message may appear. *Note that this message is for the Comm Port Wedge.* You must disable that application to free up the COM1 port before you can enable either the wedge or the scanner.

Scanner Cabling

A null modem cable is required for the Welch Allyn 1470 Imager to communicate with the 700 Series Computer when using the 700 Series Serial Cable (P/N: 226-999-001).

The Sabre 1551E / 1553 Cable connects directly to the Model 700 Comm Port.

Limitations and Capabilities

The Tethered Scanner has the following limitations:

- No auto detection of a scanner' s physical connection to COM1 port. User needs to ensure the communication settings of COM1 port matched the settings of the device.
- The Pocket PC Pocket Office applications misbehave when control characters such as carriage return are wedged. This is a known Pocket PC problem, which is being worked with Microsoft and for which a work around is being developed.
- Communications port is COM1 and cannot be changed.
- A complete bar code label is detected when the time between bytes (the inter-byte gap) exceeds 100 ms. This allows that data could be concatenated if two labels were received while the Comm Port Wedge or the 1551/1553 Tethered Scanner was not performing a read. That is, it could be wedging data just read or the read thread could be preempted. Also, the labels could appear concatenated if the scanner itself were to buffer the labels before transmitting them.

When enabled, the Comm Port Wedge menu option has the following limitation:

- There is no bar code API to get bar code data from the bar code scanner. The Comm Port Wedge transmits the data through the keyboard interface only.

When enabled, the 1551/1553 menu option has the following capabilities:

- Grid Data Editing is available.
- The source of the symbology configurations is only available via the Easy Set command labels. Only the Virtual Wedge configurations can be configured via the Data Collection Control Panel Applet Virtual Wedge page. See Appendix A, "*Control Panel Applets*," for more information.
- May transmit the data through the keyboard interface (via the Virtual Wedge).

- The bar code APIs, defined in the IADC interface, are available to get bar code data from the bar code scanner. The following example shows how to programmatically collects bar code data:

```
#include "IADC.h"                    // Linked with ITCUUID.LIB
#include "ITCAdcMgmt.h"              // Linked with ITCAdcDevMgmt.lib

   IADC* pIADC;
   HRESULT hrStatus = S_OK;

// Create a ADC COM interface to collect bar code data from the 1551E/1553
// when the 1551/1553 menu option is enabled.
   hrStatus =
   ITCDeviceOpen(TEXT("ExtScanner"),  // Name of the ADC device.
      IID_IADC,                       // COM interface to return
      ITC_DHDEVFLAG_READAHEAD,        // Device's Flags
      (LPVOID *) &pIADC);             // the returned interface

if( SUCCEEDED(hrStatus) )
   {
      BYTE byteBuffer[MAX_LABEL_SIZE];
      DWORD dwLength = 0;
   HRESULT hr = pIDC->Read(
      byteBuffer,                     // Buffer to put the ADC data.
      MAX_LABEL_SIZE,                 // Size of pDataBuffer in bytes.
      &dwLength,                      // Number bytes returned.
      NULL,                           // Time stamp of the received data. NULL.
      INFINITE                        // Number of milliseconds to wait.
   );

}
      when done using this COM interface, delete it:
ITCDeviceClose( (IUnknown **) pIADC);
```

7 Programming

The following programming information pertains to the 700 Series Color Mobile Computer:

Creating CAB Files

The Windows CE operating system uses a .CAB file to install an application on a Windows CE-based device. A .CAB file is composed of multiple files that are compressed into one file. Compressing multiple files into one file provides the following benefits:

- All application files are present.
- A partial installation is prevented.
- The application can be installed from several sources, such as a desktop computer or a Web site.

Use the CAB Wizard application (CABWIZ.EXE) to generate a .CAB file for your application.

Creating Device-Specific CAB Files

Do the following to create a device-specific .CAB file for an application, *in the order provided*:

1. Create an .INF file with Windows CE-specific modifications *(page 236)*.
2. *Optional* Create a SETUP.DLL file to provide custom control of the installation process *(page 248)*.
3. Use the CAB Wizard to create the .CAB file, using the .INF file, the optional SETUP.DLL file, and the device-specific application files as parameters *(page 249)*.

Creating an .INF File

An .INF file specifies information about an application for the CAB Wizard. Below are the sections of an .INF file:

[Version]

This specifies the creator of the file, version, and other relevant information.

Required? Yes

- **Signature**: "*signature_name*"
  Must be "$Windows NT$" as Windows CE is not available on Windows 95.
- **Provider**: "*INF_creator*"
  The company name of the application, such as "Microsoft."
- **CESignature**: "$Windows CE$"

EXAMPLE:

```
[Version]
Signature = "$Windows NT$"
Provider = "Microsoft"
CESignature = "$Windows CE$"
```

[CEStrings]

This specifies string substitutions for the application name and the default installation directory.

Required? Yes

- **AppName**: *app_name*
  Name of the application. Other instances of %AppName% in the .INF file will be replaced with this string value, such as RP32.
- **InstallDir**: *default_install_dir*
  Default installation directory on the device. Other instances of %InstallDir% in the .INF file will be replaced with this string value. Example: \storage_card\%AppName%

EXAMPLE:

```
[CEStrings]
AppName="Game Pack"
InstallDir=%CE1%\%AppName%
```

[Strings]

This section is optional and defines one or more string keys. A string key represents a string of printable characters.

Required? No

- **string_key**: *value*
  String consisting of letters, digits, or other printable characters. Enclose *value* in double quotation marks """" if the corresponding string key is used in an item that requires double quotation marks. No string_keys is okay.

EXAMPLE:

```
[Strings]
reg_path = Software\Microsoft\My Test App
```

[CEDevice]

Describes the platform for the targeted application. All keys in this section are optional. If a key is nonexistent or has no data, Windows CE does not perform any checking with the exception being *UnsupportedPlatforms*. If the *UnsupportedPlatforms* key exists but no data, the previous value is not overridden.

Required? Yes

- **ProcessorType** : *processor_type*
  The value that is returned by **SYSTEMINFO**.dwProcessorType. For example, the value for the SH3 CPU is 10003 and the MIPS CPU is 4000.
- **UnsupportedPlatforms**: *platform_family_name*
  This lists known unsupported platform family names. If the name specified in the [**CEDevice.xxx**] section is different from that in the [**CEDevice**] section, both *platform_family_name* values are unsupported for the microprocessor specified by xxx. That is, the list of unsupported platform family names is appended to the previous list of unsupported names. Application Manager will not display the application for an unsupported platform. Also, a user will be warned during the setup process if the .CAB file is copied to an unsupported device.

EXAMPLE:

```
[CEDevice]
UnsupportedPlatforms = pltfrm1 ; pltfrm1 is unsupported
[CEDevice.SH3]
UnsupportedPlatforms = ; pltfrm1 is still unsupported
```

- **VersionMin**: *minor_version*
  Numeric value returned by **OSVERSIONINFO**.dwVersionMinor. The .CAB file is valid for the currently connected device if the version of this device is greater than or equal to **VersionMin**. For Windows CE Japanese language devices, set this to 2.01
- **VersionMax**: *major_version*
  Numeric value returned by **OSVERSIONINFO**.dwVersionMajor. The .CAB file is valid for the currently connected device if the version of this device is less than or equal to **VersionMax**. For Windows CE Japanese language devices, set this to 2.01

Note: Supported Windows CE operating system versions include 1.0, 1.01, 2.0, 2.01, and 2.10. When using these numbers, be sure to include all significant digits.

- **BuildMin**: *build_number*
  Numeric value returned by **OSVERSIONINFO**.dwBuildNumber. The .CAB file is valid for the currently connected device if the version of this device is greater than or equal to **BuildMin**.
- **BuildMax**: *build_number*
  Numeric value returned by **OSVERSIONINFO**.dwBuildNumber. The .CAB file is valid for the currently connected device if the version of this device is less than or equal to **BuildMax**.

EXAMPLE:

The following code example shows three [**CEDevice**] sections: one that gives basic information for any CPU and two that are specific to the SH3 and the MIPS microprocessors.

```
[CEDevice]                      ; A "template" for all platforms
UnsupportedPlatforms = pltfrm1 ; Does not support pltfrm1

; The following specifies version 1.0 devices only.
VersionMin = 1.0
VersionMax = 1.0

[CEDevice.SH3]                  ; Inherits all [CEDevice] settings
; This will create a .CAB file specific to SH3 devices.
ProcessorType = 10003           ; SH3 .cab file is valid for SH3 microprocessors.
UnsupportedPlatforms =          ; pltfrm1 is still unsupported

; The following overrides the version settings so that no version checking is
performed.
VersionMin =
VersionMax =

[CEDevice.MIPS]                 ; Inherits all [CEDevice] settings
; This will create a .CAB file specific to "MIPS" devices.
ProcessorType = 4000            ; MIPS .CAB file is valid for MIPS microprocessor.
UnsupportedPlatforms =pltfrm2   ; pltfrm1, pltfrm2 unsupported for MIPs .CAB file.
```

Note: To create the two CPU-specific .CAB files for the SETUP.INF file in the previous example, run the CAB Wizard with the "/cpu sh3 mips" parameter.

[DefaultInstall]

This describes the default installation of your application. Note that under this section, you will list items expanded upon later in this description.

Required? Yes

- **Copyfiles**: *copyfile_list_section*
  Maps to files defined later in the .INF file, such as Files.App, Files.Font, and Files.Bitmaps.
- **AddReg**: *add_registry_section*
  Example: RegSettings.All
- **CEShortcuts**: *shortcut_list_section*
  String that identifies one more section that defines shortcuts to a file, as defined in the [**CEShortcuts**] section.
- **CESetupDLL**: *setup_DLL*
  Optimal string that specifies a SETUP.DLL file. It is written by the Independent Software Vendor (ISV) and contains customized functions for operations during installation and removal of the application. The file must be specified in the [**SourceDisksFiles**] section.
- **CESelfRegister**: *self_reg_DLL_filename*
  String that identifies files that self-register by exporting the **DllRegisterServer** and **DllUnregisterServer** Component Object Model (COM) functions. Specify these files in the [**SourceDiskFiles**] section. During installation, if installation on the device fails to call the file's exported **DllRegisterServer** function, the file's exported **DllUnregisterServer** function will not be called during removal.

EXAMPLE:

```
[DefaultInstall]
AddReg = RegSettings.All
CEShortcuts = Shortcuts.All
```

[SourceDiskNames]

This section describes the name and path of the disk on which your application resides.

Required? Yes

- **disk_ordinal**: *disk_label,,path*
  1=,"App files" , C:\Appsoft\RP32\...
  2=,"Font files",,C:\RpTools\...
  3=,"CE Tools" ,,C:\windows ce tools...
- **CESignature**: "$Windows CE$"

Example

```
[SourceDisksNames]                          ; Required section
1 = ,"Common files",,C:\app\common          ; Using an absolute path
[SourceDisksNames.SH3]
2 = ,"SH3 files",,sh3                       ; Using a relative path
[SourceDisksNames.MIPS]
2 = ,"MIPS files",,mips                     ; Using a relative path
```

[SourceDiskFiles]

This describes the name and path of the files in which your application resides.

Required? Yes

- **filename**: *disk_number[,subdir]*
  RPM.EXE = 1,c:\appsoft\...
  WCESTART.INI = 1
  RPMCE212.INI = 1
  TAHOMA.TTF = 2

Note: [**,subdir**] is relative to the location of the INF file.

Example

```
[SourceDisksFiles]      ; Required section
begin.wav = 1
end.wav = 1
sample.hlp = 1
[SourceDisksFiles.SH3]
sample.exe = 2          ; Uses the SourceDisksNames.SH3 identification of 2.
[SourceDisksFiles.MIPS]
sample.exe = 2          ; Uses the SourceDisksNames.MIPS identification of 2.
```

[DestinationDirs]

This describes the names and paths of the destination directories for the application on the target device. *Note Windows CE does not support directory identifiers.*

Required? Yes

- **file_list_section**: *0,subdir*
  String that identifies the destination directory. The following list shows the string substitutions supported by Windows CE. These can be used only for the beginning of the path. \
  %CE1% \Program Files
  %CE2% \Windows
  %CE3% \My Documents
  %CE4% \Windows\Startup
  %CE5% \My Documents
  %CE6% \Program Files\Accessories
  %CE7% \Program Files\Communication
  %CE8% \Program Files\Games
  %CE9% \Program Files\Pocket Outlook
  %CE10% \Program Files\Office
  %CE11% \Windows\Start Menu\Programs
  %CE12% \Windows\Start Menu\Programs\Accessories
  %CE13% \Windows\Start Menu\Programs\Communications
  %CE14% \Windows\Start Menu\Programs\Games
  %CE15% \Windows\Fonts
  %CE16% \Windows\Recent
  %CE17% \Windows\Start Menu
  %InstallDir%
  Contains the path to the target directory selected during installation. It is declared in the [**CEStrings**] section
  %AppName%
  Contains the application name defined in the [**CEStrings**] section.

Example

```
[DestinationDirs]
Files.Common  = 0,%CE1%\My Subdir    ; \Program Files\My Subdir
Files.Shared  = 0,%CE2%              ; \Windows
```

[CopyFiles]

This section, under the [**DefaultInstall**] section, describes the default files to copy to the target device. Within the [**DefaultInstall**] section, files were listed that must be defined elsewhere in the INF file. This section identifies that mapping and may contain flags.

Required? Yes

- **copyfile_list_section**: *destination_filename,[source_filename]*
  The *source_filename* parameter is optional if it is the same as *destination_filename*.
- **copyfile_list_section**: *flags*
  The numeric value that specifies an action to be done while copying files. The following table shows values supported by Windows CE.

Flag	Value	Description
COPYFLG_WARN_IF_SKIP	0x00000001	Warn user if skipping a file is attempted after error.
COPYFLG_NOSKIP	0x00000002	Do not allow a user to skip copying a file.
COPYFLG_NO_OVERWRITE	0x00000010	Do not overwrite files in destination directory.
COPYFLG_REPLACEONLY	0x00000400	Copy the source file to the destination directory only if the file is already in the destination directory.
CE_COPYFLG_NO_DATE_DIALOG	0x20000000	Do not copy files if the target file is newer.
CE_COPYFLG_NODATECHECK	0x40000000	Ignore date while overwriting the target file.
CE_COPYFLG_SHARED	0x80000000	Create a reference when a shared DLL is counted.

Example

```
[DefaultInstall.SH3]
CopyFiles = Files.Common, Files.SH3
[DefaultInstall.MIPS]
CopyFiles = Files.Common, Files.MIPS
```

[AddReg]

This section, under the [**DefaultInstall**] section, is optional and describes the keys and values that the .CAB file adds to the device registry. Within the [**DefaultInstall**] section, a reference may have been made to this section, such as "AddReg=RegSettings.All". This section defines the options for that setting.

Required? No

- **add_registry_section**: *registry_root_string*
  String that specifies the registry root location. The following list shows the values supported by Windows CE.
  - HKCR Same as HKEY_CLASSES_ROOT
  - HKCU Same as HKEY_CURRENT_USER
  - HKLM Same as HKEY_LOCAL_MACHINE
- **add_registry_section**: *value_name*
  Registry value name. If empty, the "default" registry value name is used.
- **add_registry_section**: *flags*
  Numeric value that specifies information about the registry key. The following table shows the values that are supported by Window CE.

Flag	Value	Description
FLG_ADDREG_NOCLOBBER	0x00000002	If the registry key exists, do not overwrite it. Can be used with any of the other flags in this table.
FLG_ADDREG_TYPE_SZ	0x00000000	REG_SZ registry data type.
FLG_ADDREG_TYPE_MULTI_SZ	0x00010000	REG_MULTI_SZ registry data type. Value field that follows can be a list of strings separated by commas.
FLG_ADDREG_TYPE_BINARY	0x00000001	REG_BINARY registry data type. Value field that follows must be a list of numeric values separated by commas, one byte per field, and must not use the 0x hexadecimal prefix.
FLG_ADDREG_TYPE_DWORD	0x00010001	REG_DWORD data type. The noncompatible format in the Win32 Setup .INF documentation is supported.

Example

```
AddReg = RegSettings.All

[RegSettings.All]
HKLM,%reg_path%,,0x00000000,alpha           ; <default> = "alpha"
HKLM,%reg_path%,test,0x00010001,3           ; Test = 3
HKLM,%reg_path%\new,another,0x00010001,6    ; New\another = 6
```

[CEShortCuts]

This section, a Windows CE-specific section under the [**DefaultInstall**] section, is optional and describes the shortcuts that the installation application creates on the device. Within the [**DefaultInstall**] section, a reference may have been made to this section, such as "ShortCuts.All". This section defines the options for that setting.

Required? No

- **shortcut_list_section**: *shortcut_filename*
  String that identifies the shortcut name. It does not require the .LNK extension.
- **shortcut_list_section**: *shortcut_type_flag*
  Numeric value. Zero or empty represents a shortcut to a file; any non-zero numeric value represents a shortcut to a folder.
- **shortcut_list_section**: *target_file_path*
  String value that specifies the destination location. Use the target file name for a file, such as MyApp.exe, that must be defined in a file copy list. For a path, use a *file_list_section* name defined in the [**DestinationDirs**] section, such as *DefaultDestDir*, or the *%InstallDir%* string.
- **shortcut_list_section**: *standard_destination_path*
  Optional string value. A standard *%CEx%* path or *%InstallDir%*. If no value is specified, the *shortcut_list_section* name of the current section or the *DefaultDestDir* value from the [**DestinationDirs**] section is used.

Example

```
CEShortcuts = Shortcuts.All
[Shortcuts.All]
Sample App,0,sample.exe        ; Uses the path in DestinationDirs. Sample
App,0,sample.exe,%InstallDir%  ; The path is explicitly specified.
```

Sample .INF File

```
[Version]          ; Required section
Signature = "$Windows NT$"
Provider = "Intermec Technologies Corporation"
CESignature = "$Windows CE$"

;[CEDevice]
;ProcessorType =

[DefaultInstall]   ; Required section
CopyFiles = Files.App, Files.Fonts, Files.BitMaps, Files.Intl,
Files.TelecomNcsCE, Files.Windows, Files.Import, Files.Export, Files.Work,
Files.Database, Files.WinCE AddReg = RegSettings.All ;CEShortcuts =
Shortcuts.All

[SourceDisksNames] ; Required section
1 = ,"App files" ,,c:\appsoft\...
2 = ,"Font files" ,,c:\WinNT\Fonts
3 = ,"CE Tools"  ,,c:\windows ce tools\wce212\6110ie\mfc\lib\x86

[SourceDisksFiles] ; Required section
rpm.exe   = 1,C:\Appsoft\program\wce212\WCEX86Rel6110
wcestart.ini = 1
```

```
rpmce212.ini = 1
intermec.bmp = 1
rpmlogo.bmp = 1
rpmname.bmp = 1
import.bmp  = 1
export.bmp  = 1
clock.bmp  = 1
printer.bmp = 1
filecopy.bmp = 1
readme.txt  = 1
lang_eng.bin = 1
rpmdata.dbd = 1,database\wce1
tahoma.ttf  = 2
mfcce212.dll = 3
olece212.dll = 3
olece211.dll = 1,c:\windows ce tools\wce211\NMSD61102.11\mfc\lib\x86
rdm45wce.dll = 1,c:\rptools\rdm45wce\4_50\lib\wce212\wcex86rel
picfmt.dll  = 1,c:\rptools\picfmt\1_00\wce212\wcex86rel6110
fmtctrl.dll = 1,c:\rptools\fmtctrl\1_00\wce212\wcex86rel6110
ugrid.dll  = 1,c:\rptools\ugrid\1_00\wce212\wcex86rel6110
simple.dll  = 1,c:\rptools\pspbm0c\1_00\wce211\wcex86rel
psink.dll = 1,c:\rptools\psink\1_00\wce211\WCEX86RelMinDependency
pslpwce.dll =1,c:\rptools\pslpm0c\1_00\wce211\WCEX86RelMinDependency
npcpport.dll = 1,c:\rptools\cedk\212_03\installable drivers\printer\npcp
;dexcom.dll  = 1,c:\rptools\psdxm0c\1_00\x86
ncsce.exe  = 1,c:\rptools\ncsce\1_04
nrinet.dll  = 1,c:\rptools\ncsce\1_04

[DestinationDirs]  ; Required section
;Shortcuts.All = 0,%CE3%  ; \Windows\Desktop
Files.App       = 0,%InstallDir%
Files.DataBase     = 0,%InstallDir%\DataBase
Files.BitMaps      = 0,%InstallDir%\Bitmaps
Files.Fonts        = 0,%InstallDir%\Fonts
Files.Intl         = 0,%InstallDir%\Intl
Files.TelecomNcsCE = 0,%InstallDir%\Telecom\NcsCE
Files.Windows      = 0,%InstallDir%\Windows
Files.Import       = 0,%InstallDir%\Import
Files.Export       = 0,%InstallDir%\Export
Files.Work         = 0,%InstallDir%\Work
Files.WinCE        = 0,\storage_card\wince

[CEStrings]        ; Required section
AppName = Rp32
InstallDir = \storage_card\%AppName%

[Strings]       ; Optional section
;[Shortcuts.All]
;Sample App,0,sample.exe                ; Uses the path in DestinationDirs.
;Sample App,0,sample.exe,%InstallDir%   ; The path is explicitly specified.

[Files.App]
rpm.exe,,,0
rpm.ini,rpmce212.ini,,0
mfcce212.dll,,,0
olece212.dll,,,0
olece211.dll,,,0
rdm45wce.dll,,,0
picfmt.dll,,,0
```

```
fmtctrl.dll,,,0
ugrid.dll,,,0
simple.dll,,,0
psink.dll,,,0
pslpwce.dll,,,0
npcpport.dll,,,0
;dexcom.dll,,,0

[Files.DataBase]
rpmdata.dbd,,,0

[Files.Fonts]
tahoma.ttf,,,0

[Files.BitMaps]
intermec.bmp,,,0
rpmlogo.bmp,,,0
rpmname.bmp,,,0
import.bmp,,,0
export.bmp,,,0
clock.bmp,,,0
printer.bmp,,,0
filecopy.bmp,,,0

[Files.Intl]
lang_eng.bin,,,0

[Files.TelecomNcsCE]
ncsce.exe,,,0
nrinet.dll,,,0

[Files.Windows]
readme.txt,,,0

[Files.Import]
readme.txt,,,0

[Files.Export]
readme.txt,,,0

[Files.Work]
readme.txt,,,0

[Files.WinCE]
wcestart.ini,,,0

[RegSettings.All]
HKLM,"SOFTWARE\Microsoft\Shell\AutoHide",,0x00010001,1
; Autohide the taskbar HKLM,"SOFTWARE\Microsoft\Shell\OnTop",,0x00010001,0
  ; Shell is not on top HKLM,"SOFTWARE\Microsoft\Clock",SHOW_CLOCK,0x00010001,0
; Clock is not on taskbar
```

Using Installation Functions in SETUP.DLL

SETUP.DLL is an optional file that enables you to perform custom operations during installation and removal of your application. The following list shows the functions that are exported by SETUP.DLL.

- **Install_Init**
  Called before installation begins. Use this function to check the application version when reinstalling an application and to determine if a dependent application is present.
- **Install_Exit**
  Called after installation is complete. Use this function to handle errors that occur during application installation.
- **Uninstall_Init**
  Called before the removal process begins. Use this function to close the application, if the application is running.
- **Uninstall_Exit**
  Called after the removal process is complete. Use this function to save database information to a file and delete the database and to tell the user where the user data files are stored and how to reinstall the application.

Note; Use [**DefaultInstall**] → **CESelfRegister** (page 240) in the .INF file to point to SETUP.DLL.

After the CAB File Extraction

Cab files that need to cause a warm reset after cab extraction will need to create the __RESETMEPLEASE__.TXT file in the "\Windows" directory. The preferred method to create this file is within the DllMain portion of the SETUP.DLL file. It looks like this:

```
BOOL APIENTRY DllMain( HANDLE hModule, DWORD  ul_reason_for_call, LPVOID
lpReserved  )
{
  switch (ul_reason_for_call)
    {
      case DLL_PROCESS_ATTACH:
        break;
      case DLL_THREAD_ATTACH:
        break;
      case DLL_THREAD_DETACH:
        break;
      case DLL_PROCESS_DETACH:
        if (bInstallSuccessful) {
          HANDLE h;
          h = CreateFile(L"\\Windows\\__resetmeplease__.txt",
            GENERIC_READ|GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
            FILE_ATTRIBUTE_HIDDEN, NULL);
          if (h != INVALID_HANDLE_VALUE)
            CloseHandle(h);
        }
        break;
  }
  return TRUE;
}
```

The system software looks for the following directory structure and files on the installed media card whether it be an SD card or CF card or embedded flash file system. No other folders need exist.

```
\2577\autorun.exe
\2577\autorun.dat
\2577\autocab.exe
\2577\autocab.dat
\cabfiles\*.cab
```

Creating CAB Files with CAB Wizard

After you create the .INF file and the optional SETUP.DLL file, use the CAB Wizard to create the .CAB file. The command-line syntax for the CAB Wizard is as follows:

```
cabwiz.exe "inf_file" [/dest dest_directory] [/err error_file] [/cpu cpu_type
[cpu_type]]
```

A batch file, located in <program> directory, with the following commands, works well:

```
cd\"Windows CE Tools"\WCE211\"MS HPC Pro"\support\appinst\bin
cabwiz.exe c:\appsoft\<program>\<inf_file_name>
cd \appsoft\<program>
```

- "inf_file"
  The SETUP.INF file path.
- *dest_directory*
  The destination directory for the .CAB files. If no directory is specified, the .CAB files are created in the "inf_file" directory.
- *error_file*
  The file name for a log file that contains all warnings and errors that are encountered when the .CAB files are compiled. If no file name is specified, errors are displayed in message boxes. If a file name is used, the CAB Wizard runs without the user interface (UI); this is useful for automated builds.
- *cpu_type*
  Creates a .CAB file for each specified microprocessor tag. A microprocessor tag is a label used in the Win32 SETUP.INF file to differentiate between different microprocessor types. The */cpu* parameter, followed by multiple *cpu_type* values, must be the last qualifier in the command line.

Example

This example creates .CAB files for the SH3 and MIPS microprocessors, assuming that the Win32 SETUP.INF file contains the SH3 and MIPS tags:

```
cabwiz.exe "c:\myfile.inf" /err myfile.err /cpu sh3 mips
```

Note: CABWIZ.EXE, MAKECAB.EXE, and CABWIZ.DDF (Windows CE files available on the Windows CE Toolkit) must be installed in the same directory on the desktop computer. Call CABWIZ.EXE using its full path for the CAB Wizard application to run correctly.

Troubleshooting the CAB Wizard

To identify and avoid problems that might occur when using the CAB Wizard, follow these guidelines:

- Use %% for a percent sign (%) character when using this character in an .INF file string, as specified in Win32 documentation. This will not work under the [**Strings**] section.
- Do not use .INF or .CAB files created for Windows CE to install applications on Windows-based desktop platforms.
- Ensure the MAKECAB.EXE and CABWIZ.DDF files, included with Windows CE, are in the same directory as CABWIZ.EXE.
- Use the full path to call CABWIZ.EXE.
- Do not create a .CAB file with the MAKECAB.EXE file included with Windows CE. You must use CABWIZ.EXE, which uses MAKECAB.EXE to generate the .CAB files for Windows CE.
- Do *not* set the read-only attribute for .CAB files.

FTP Server

FTP support is provided through the FTP Server application FTPDCE.EXE (MS Windows CE Versions) which is provided as part the base system.

FTPDCE is the Internet File Transfer Protocol (FTP) server process. The server can be invoked from an application or command line. Besides servicing FTP client requests the FTP Server also send a "network announcement" to notify prospective clients of server availability.

Synopsis

ftpdce [*options*]

Options

- *-Aaddr*
  Sets the single target address to which to send the network announcement. *Default is broadcast.*
- *-Bbyte*
  Sets the FTP data block size. Smaller sizes may be useful over slower links. *Default is 65536.*
- *-Cname*
  Sets the device name. Used by Intermec management software.
- *-Fvalue*
  Disables the default Intermec account. A value of "0" disables the account. *Default is "1".*

Note: Disabling the default account without providing a working access control list on the server will result in a device that will not accept any FTP connections.

- *-Hsec*
  Sets the interval between network announcements in seconds.A value of "0" turns the network announcement off. *Default is 30 seconds.*
- *-Iip*
  Sets the preferred 6920 Communications Server *(optional).*
- *-Llog*
  Sets the state of logging. *Default is 0 (disabled).*
- *-Nsec*
  Specifies the number of seconds to wait before starting FTP server services.
- *-Pport*
  Sets the UDP port on which the network announcement will be sent. *Default port is 52401.*
- *-Qport*
  Sets the port on which the FTP Server will listen for connections. *Default port is 21.*

- *-Rdir*
  Sets the FTP mount point to this directory. Default is the rootdirectory of the drive from which the FTP Server program was executed.
- *-Tscript*
  Sets the script name for the 6920 Communications Server to process.
- *-Uurl*
  Sets the default URL for this device.
- *-Z"parms"*
  Sets extended parameters to be included in the network announcement.

Configurable Parameters Via the Registry Editor

The following parameters receive default values during the installation of the Intermec FTP Server components. A few of the parameters are visible in the registry by default, but most must be created in order to modify the default behavior of the FTP server.

BlockSize

Setting this parameter forces the Intermec FTP Server to transmit and receive Ethernet packets using the specified data block size. By default, the FTP server transmits and receives data using a 64K data block size. Adjusting this value may be useful in certain wireless TCP/IP installations.

Key

`HKLM\Software\Intermec\IFTP`

Value Type

REG_DWORD - data block size, in bytes.

Valid Range

0x100-0x10000 (256-65536 decimal).

Default

65536

DeviceName
This parameter forces the Intermec FTP Server to include the specified device name in the Intermec Device Network Announcement (IDNA). Adjusting this value may be useful in assigning a symbolic name to this device for asset tracking.

Key
`HKLM\Software\Intermec\IFTP`

Value Type
REG_SZ

Valid Range
None.

Default
None.

DeviceURL
This parameter forces the Intermec FTP Server to transmit the specified URL in the IDNA. This can be used by Intermec management software for asset management.

Key
`HKLM\Software\Intermec\IFTP`

Value Type
REG_SZ

Valid Range
None.

Default
None.

IDNATarget
This parameter forces the Intermec FTP Server to transmit the IDNA to a specific destination instead of a general UDP broadcast. This parameter is useful on networks that do not allow UDP broadcasts to be routed between subnets. The use of this parameter will restrict the reception of the IDNA to the target destination only.

Key
`HKLM\Software\Intermec\IFTP`

Value Type
REG_SZ

Valid Range
None.

Default
None.

ManifestName
This parameter forces the Intermec FTP Server to transmit the specified manifest name in the IDNA. This parameter is used by the Intermec 6920 Communications Server for communication transactions. See the 6920 Communications Server documentation for proper use of this parameter.

Key
`HKLM\Software\Intermec\IFTP`

Value Type
REG_SZ

Valid Range
None.

Default
iftp.ini

PauseAtStartup
This parameter forces the Intermec FTP Server to sleep for the specified number of seconds before making the FTP service available on the device.

Key
`HKLM\Software\Intermec\IFTP`

Value Type
REG_DWORD - stored in seconds.

Valid Range
None.

Default
0

Root
This parameter forces the Intermec FTP Server to set the root of the FTP mount point to the specified value. *Note that this must map to an existing directory or you will not be able to log into the FTP Server.*

Key
`HKLM\Software\Intermec\IFTP`

Value Type
REG_SZ

Valid Range
None.

Default
\

Transferring Files Over TCP/IP Networks

The File Transfer Protocol (FTP) server transfers files over TCP/IP networks. The FTPDCE.EXE program is a version that does not display a window, but can run in the background.

FTPDCE is the Internet File Transfer Protocol (FTP) server process. The server can be invoked from an application or command line. Besides servicing FTP client requests, the FTP Server also sends a "network announcement" to notify prospective clients of server availability.

Remarks

The FTP Server currently supports the following FTP requests:

- **CDUP**
  Changes to the parent directory of the current working directory.
- **CWD**
  Changes working directory.
- **DELE**
  Deletes a file.
- **HELP**
  Gives help information.
- **LIST** *(This FTP request is the same as the ls -lgA command).*
  Gives list files in a directory.
- **MKD**
  Makes a directory.
- **MODE** *(Always Uses Binary)*.
  Specifies data transfer mode.
- **NLST**
  Gives a name list of files in directory (this FTP request is the same as the *ls* command).
- **NOOP**
  Does nothing.
- **PASS**
  Specifies a password.
- **PWD**
  Prints the current working directory.
- **QUIT**
  Terminates session.
- **RETR**
  Retrieves a file.
- **RMD**
  Removes a directory.
- **RNFR**
  Specifies rename-from file name.

- **RNTO**
  Specifies rename-to file name.
- **STOR**
  Stores a file.
- **SYST**
  Shows the operating system type of server system.
- **TYPE** *(Binary transfers only.)*
  Specifies the data transfer type with the Type parameter.
- **USER**
  Specifies user name.
- **XCUP** *(Not Normally Used)*
  Changes the parent directory of the current working directory.
- **XCWD** *(Not Normally Used)*
  Changes the current directory.
- **XMKD** *(Not Normally Used)*
  Creates a directory.
- **XPWD** *(Not Normally Used)*
  Prints the current working directory.
- **XRMD** *(Not Normally Used)*
  Removes a directory.
- **SITE**
  The following nonstandard or operating system (OS)-specific commands are supported by the SITE request. For Microsoft FTP clients, you can send site commands by preceding the command with "quote" such as "quote site status."

  - **ATTRIB**
    Gets or sets the attributes of a given file. (SITE ATTRIB)

    Usage: **QUOTE SITE ATTRIB** [*+R* | *-R*] [*+A* | *-A*] [*+S* | *-S*] [*+H* | *-H*] [[*path*] *filename*]

+	Sets an attribute.
`	Clears an attribute.
R	Read-only file attribute.
A	Archive file attribute.
S	System file attribute.
H	Hidden file attribute.

    To retrieve the attributes of a file, only specify the file. The server response will be: *200-AD SHRCEIX filename*

    If the flag exists in its position shown above, it is set. Also, in addition to the values defined above, there is also defined:

C	Compressed file attribute.
E	Encrypted file attribute.
I	INROM file attribute.
X	XIP file attribute (execute in ROM, not shadowed in RAM).

- **BOOT**
  Reboots the server OS. This will cause the system on which the server is executing to reboot. The FTP Server will shut down cleanly before reboot. All client connections will be terminated. Cold boot is default except for the PocketPC build in which the default is warm boot.
  (SITE BOOT)

  Usage: **QUOTE SITE BOOT** [*WARM* | *COLD*]
- **COPY**
  Copies a file from one location to another. (SITE COPY)

  Usage: **QUOTE SITE COPY** [*source*] [*destination*]

Example

```
QUOTE SITE COPY '\Storage Card\one.dat' '\Storage
Card\two.dat'
```

- **EXIT**
  Exits the FTP Server. This command will shut down the FTP Server thus terminating all client connections. (SITE EXIT)

  Usage: **QUOTE SITE EXIT**
- **HELP**
  Gives site command help information. (SITE HELP)

  Usage: **QUOTE SITE HELP** [*command*]
- **KILL**
  Terminates a running program. (SITE KILL)

  Usage: **QUOTE SITE KILL** [*program* | *pid*]
- **LOG**
  Opens or closes the program log. (SITE LOG)

  Usage: **QUOTE SITE LOG** [*open* [*filename*]| *close*]
- **PLIST**
  Lists the running processes *(not supported on all platforms)*.
  (SITE PLIST)

  Usage: **QUOTE SITE PLIST**
- **RUN**
  Starts a program running. If the program to run has spaces in path or filename, wrapping the name with single quotes is required.

  Usage: **QUOTE SITE RUN** [*program*]

Example

```
QUOTE SITE RUN '\Storage Card\app.exe'
```

- **STATUS**
  Returns the current settings of the FTP Server. MAC, serial number, model, IP address, network announcement information as well as OS memory usage are returned. (SITE STATUS)

  Usage: **QUOTE SITE STATUS**

- **TIMEOUT**
  Toggles idle timeout between 120 to 1200 seconds (2 to 20 minutes). If this timer expires with no activity between the client and the server, the client connection will be disconnected. If the optional seconds argument is supplied, the server will set the connection timeout to the number of seconds specified. *Default is 120 seconds or 2 minutes.* (SITE TIMEOUT)

  Usage: **QUOTE SITE TIMEOUT** [*seconds*]

The remaining FTP requests specified in RFC 959 are recognized, but not implemented.

The banner returned in the parenthetical portion of its greeting shows the version number of the FTP Server as well as the MAC address, serial number and OS of the machine hosting the server.

The FTP Server supports browsing from the latest Netscape and Microsoft web browsers. Drag-and-drop capability is available using this environment.

The FTPDCMDS subdirectory contains commands that can be used from the web browser.

- Click EXITME.BIN to execute a SITE EXIT command.
- Click REBOOTME.BIN to execute SITE BOOT command.
- Use the GET command on these files to have the FTP Server execute these commands.

  - **Security**:
    A customer configurable access control list may be installed on the 700 Series Computer. This list will allow customers to restrict access via the FTP Server to the users they wish. This is in addition to the default Intermec account which can be disabled using the *-F0* option at runtime.

    The access control list is named FTPDCE.TXT and is placed in the same directory on the 700 Series Computer as the FTPDCE.EXE server. The FTP Server will encrypt this file to keep the information safe from unauthorized users. This file is encrypted when the FTP Server is started so a file that is placed onto the 700 Series Computer after the FTP Server starts will require a restart of the FTP Server to take effect.

    The format of the FTPDCE.TXT is as follows:

```
FTPDCE:user1!passwd1<cr><lf>user2!passwd2<cr><lf>user3!passw
d3<cr><lf>...
```

Note: The user accounts and passwords are case sensitive.
Once the access control list is encrypted on the 700 Series Computer, the FTP Server will hide this file from users. Once an access control list has been installed on the 700 Series Computer, a new one will not be accepted by the FTP Server until the previous one is removed.
Encrypted access control lists are not portable between 700 Series Computers.

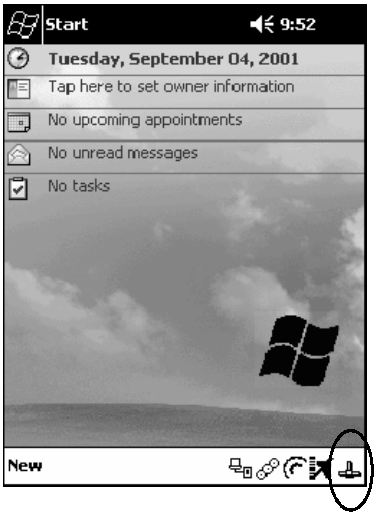
Stopping the FTP Server from Your Application

To allow application programmers the ability to programmatically shut down the FTP Server, the FTP Server periodically tests to see if a named event is signaled. The name for this event is "ITC_IFTP_STOP" (no quotes).

For examples on how to use this event, consult the Microsoft Developer Network Library at *http://www.msdn.com*. The MSDN Library is an essential resource for developers using Microsoft tools, products, and technologies. It contains a bounty of technical programming information, including sample code, documentation, technical articles, and reference guides.

Autostart FTP

This automatically starts the FTP Server (FTPDCE.EXE) when the 700 Series Computer is powered on. This is provided with the NDISTRAY program, which displays the popup menu that currently allows you to load and unload the network drivers. Tap the antenna icon in the System Tray of the Today screen *(a sample antenna icon is circled below)* to get this popup menu.



The default is to start the FTP Server at boot time, unless the following registry entry is defined and set to "0" which disables AutoFTP. "1" enables the AutoFTP. The entry can be set from the NDISTRAY pop-up menu by selecting either **AutoFTP On** or **AutoFTP Off**.

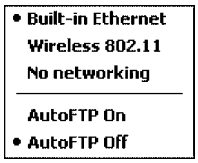
```
HKEY_LOCAL_MACHINE\Software\Intermec\Ndistray\StartupIFTP
```

These new entries are located below the selections to load the network drivers. If the StartupIFTP registry key is not defined, the FTP Server is loaded by default, to provide "out-of-the-box" capability for customers who want to begin loading files to the 700 Series Computer without any prior configuration.

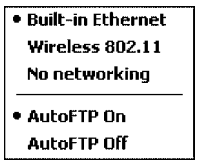
Note: If a network driver is unloaded using the NDISTRAY popup menu, and the FTP Server is running, the FTP Server is stopped.

On a resume, if AutoFTP is enabled and the FTP Server is running, it is stopped and restarted. NDISTRAY uses a helper application named RESETIFTP to implement the restart on resume feature. To do an AutoFTP Installation Check:

1. Ensure the FTP Server is running "out-of-the-box" the first time.
2. Tap **Start → Today** to access the Today screen, then tap the antenna icon in the System Tray to bring up the NDISTRAY pop-up menu. Select **AutoFTP Off** to disable AutoFTP. Do a warm boot and confirm the FTP Server is not running.



3. Tap **Start → Today** to access the Today screen, then tap the antenna icon in the System Tray to bring up the NDISTRAY pop-up menu. Select **AutoFTP On** to enable AutoFTP, reboot and confirm it is running.



4. Unload the network driver when the FTP Server is running and confirm that it is not running any more.
5. Load the FTP Server, establish a connection, then suspend and resume. The server should still be running, but the FTP connection to the client should be dropped.

Full Screen

Pocket PC is a hardware specification created by Microsoft Corporation. Devices that wish to carry the Pocket PC logo must meet the minimum hardware requirements set in the Pocket PC specification. Manufacturers are free to add extra hardware functionality.

Pocket PC 2002 devices also use a specialized version of the CE operating system. This OS is built from Windows CE 3.0 but contains customizations, most notably the lack of a desktop and the addition of the Today Screen.

To carry the Pocket PC logo, all devices must be tested at an Independent Test Laboratory. The ITL testing is done based on Microsoft requirements. The test lab then reports the findings back to Microsoft Corporation and Intermec Technologies. If the 700 Series Computer passed all tests, Intermec is allowed to ship the device with the Pocket PC logo. Each time the operating system is modified, Intermec must resubmit to ITL testing.

This means we cannot change the operating system much and still be a Pocket PC device. For example, if we remove Word from the Start menu, the device would fail ITL testing and we would not be able to ship devices with the Pocket PC logo.

Although many customers want a Pocket PC device, some customers would prefer that their users not have access to all of the Pocket PC features. Intermec cannot customize the operating system in any way but a custom application can:

- Delete items from the Start menu, and Programs folder. These items are just shortcuts in the file system so the application is not really being deleted. Cold booting the device will bring these items back so the application will need to be run on every cold boot.
- Use the RegFlushKey() API to save a copy of the registry to a storage device. See the *Recovery CD Help* for more information on how to do this. Saving a copy of the registry will allow most system settings to be restored in a cold boot situation.
- Use the SHFullScreen() API in conjunction with other APIs to make the application take up the entire display and prevent the start menu from being available.
- Remap keys and disable keys on the keypad.
- Create a custom SIP.
- Make changes to the registry to configure the device.

Should you want your 700 Series Computer to display a full screen, keep in mind that your computer is Pocket-PC certified by Microsoft Corporation. Check out resources on programming for the Pocket PC, using the following links. These instructions give full instructions on how to display full screen.

- Instructions on how to create a full screen application for eVC++ applications using an SHFullScreen() API:
  http://support.microsoft.com/support/kb/articles/Q266/2/44.ASP
- Instructions on how to create a full screen application for eVB applications also using the SHFullScreen() API:
  http://support.microsoft.com/support/kb/articles/Q265/4/51.ASP

Kernel I/O Controls

This describes the KernelIoControl() functions available to application programmers. Most C++ applications will need to prototype the function as the following to avoid link and compile errors.

```
extern "C" BOOL KernelIoControl(DWORD dwIoControlCode, LPVOID lpInBuf, DWORD
nInBufSize, LPVOID lpOutBuf, DWORD nOutBufSize, LPDWORD lpBytesReturned);
```

IOCTL_HAL_GET_DEVICE_INFO

This IOCTL returns either the platform type or the OEMPLATFORM name based on an input value.

Syntax

```
BOOL KernelIoControl( IOCTL_HAL_GET_DEVICE_INFO, LPVOID
lpInBuf, DWORD nInBufSize, LPVOID lpOutBuf, DWORD
nOutBufSize, LPDWORD lpBytesReturned );
```

Parameters

lpInBuf — Points to a DWORD containing either the SPI_GETPLATFORMTYPE or SPI_GETOEMINFO value.

lpInBufSize — Must be set to sizeof(DWORD).

lpOutBuf — Must point to a buffer large enough to hold the return data of the function. If SPI_GETPLATFORMTYPE is specified in *lpInBuf*, then the "PocketPC\0" Unicode string is returned. If SPI_GETOEMINFO is specified in *lpInBuf*, then the "Intermec 700\0" Unicode string is returned.

nOutBufSize — The size of *lpOutBuf* in bytes. Must be large enough to hold the string returned.

lpBytesReturned — The actual number of bytes returned by the function for the data requested.

Return Values

Returns TRUE if function succeeds. Returns FALSE if the function fails. GetLastError() may be used to get the extended error value.

IOCTL_HAL_ITC_READ_PARM

Usage

#include “oemioctl.h”

Syntax

```
BOOL KernelIoControl( IOCTL_HAL_ITC_READ_PARM,LPVOID
lpInBuf,DWORD nInBufSize,LPVOID lpOutBuf,DWORD
nOutBufSize,LPDWORD lpBytesReturned );
```

Parameters

lpInBuf Points to this structure. See “*ID Field Values*” below.

```
struct PARMS {
  BYTE  id;
  BYTE  ClassId;
};
```

nInBufSize Must be set to the size of the PARMS structure.

lpOutBuf Must point to a buffer large enough to hold the return data of the function. If this field is set to NULL and *nOutBufSize* is set to zero when the function is called the function will return the number bytes required by the buffer.

nOutBufSize The size of *lpOutBuf* in bytes.

lpBytesReturned The number of bytes returned by the function for the data requested.

Return Values

Returns TRUE if function succeeds. Returns FALSE if the function fails. GetLastError() may be used to get the error value. Either ERROR_INVALID_PARAMETER or ERROR_INSUFFICIENT_BUFFER may be returned when this function is used to get the error.

ID Field Values

The *id* field of the PARMS structure may be one of the following values:

- **ITC_NVPARM_ETHERNET_ID**
  This IOCTL returns the Ethernet 802.11 MAC Address. Six bytes are returned in the buffer pointed to by the *lpOutBuffer* parameter.
- **ITC_NVPARM_SERIAL_NUM**
  This IOCTL returns the serial number of the device in BCD format. Six bytes are returned in the buffer pointed to by the *lpOutBuffer* parameter.
- **ITC_NVPARM_MANF_DATE**
  This IOCTL returns the device date of manufacture in the BCD YYYY/MM/DD format. Four bytes are returned in the buffer pointed to by the *lpOutBuffer* parameter.

- **ITC_NVPARM_SERVICE_DATE**
  This IOCTL returns the device's date of last service in BCD YYYY/MM/DD format. Four bytes are returned in the buffer pointed to by the *lpOutBuffer* parameter.
- **ITC_NVPARM_DISPLAY_TYPE**
  This IOCTL returns the device's display type. One byte is returned in the buffer pointed to by the *lpOutBuffer* parameter.
- **ITC_NVPARM_EDG_IP**
  This IOCTL returns the device Ethernet debug IP address. Four bytes are returned in the buffer pointed to by the *lpOutBuffer* parameter.
- **ITC_NVPARM_EDBG_SUBNET**
  This IOCTL returns the device Ethernet debug subnet mask. Four bytes are returned in the buffer pointed to by the *lpOutBuffer* parameter.
- **ITC_NVPARM_ECN**
  This IOCTL returns ECNs applied to the device in a bit array format. Four bytes are returned in the buffer pointed to by the *lpOutBuffer* parameter.
- **ITC_NVPARM_CONTRAST**
  This IOCTL returns the device default contrast setting. Two bytes are returned in the buffer pointed to by the *lpOutBuffer* parameter.
- **ITC_NVPARM_MCODE**
  This IOCTL returns the manufacturing configuration code for the device. Sixteen bytes are returned in the buffer pointed to by the *lpOutBuffer* parameter.
- **ITC_NVPARM_VERSION_NUMBER**
  This IOCTL returns the firmware version for various system components. These values for the *ClassId* field of the PARMS structure are allowed when ITC_NVPARM_VERSION_NUMBER is used in the *id* field:
  - **VN_CLASS_KBD**
    Returns a five-byte string, including null terminator, that contains an ASCII value which represents the keyboard microprocessor version in the system. The format of the string is *x.xx* with a terminating null character.
  - **VN_CLASS_ASIC**
    Returns a five-byte string, including null terminator, that contains an ASCII value which represents the version of the FPGA firmware in the system. The format of the string is *x.xx* with a terminating null character.
  - **VN_CLASS_BOOTSTRAP**
    Returns a five-byte string, including null terminator, that contains an ASCII value which represents the version of the Bootstrap Loader firmware in the system. The format of the string is *x.xx* with a terminating null character.

- **ITC_NVPARM_INTERMEC_SOFTWARE_CONTENT**
  This IOCTL reads the manufacturing flag bits from the non-volatile data store that dictates certain software parameters. A BOOLEAN DWORD is returned in the buffer pointed to by *lpOutBuffer* that indicates if Intermec Content is enabled in the XIP regions. TRUE indicates that it is enabled. FALSE indicates that it is not enabled.
- **ITC_NVPARM_ANTENNA_DIVERSITY**
  This IOCTL reads the state of the antenna diversity flag. A BOOLEAN DWORD is returned in the buffer pointed to by *lpOutBuffer* that indicates if there is a diversity antenna installed. TRUE indicates that it is installed. FALSE indicates that it is not installed.
- **ITC_NVPARM_WAN_RI**
  This IOCTL reads the state of the WAN ring indicator flag. A BOOLEAN DWORD is returned in the buffer pointed to by *lpOutBuffer* that indicates the polarity of the WAN RI signal. TRUE indicates active high. FALSE indicates active low.
- **ITC_NVPARM_RTC_RESTORE**
  This IOCTL reads the state of the real-time clock restore flag. A BOOLEAN DWORD is returned in the buffer pointed to by *lpOutBuffer*. TRUE indicates that the RTC will be restored upon a cold boot. FALSE indicates that the RTC will not be restored.
- **ITC_NVPARM_INTERMEC_DATACOLLECTION_SW**
  This IOCTL reads the state of the data collection software enabled flag. A BOOLEAN DWORD is returned in the buffer pointer to by *lpOutBuffer* that indicates the data collection software is to be installed at boot time. FALSE indicates the data collection software should not be installed.
- **ITC_NVPARM_INTERMEC_DATACOLLECTION_HW**
  This IOCTL reads the data collection hardware flags. A BYTE is returned in the buffer pointer to by *lpOutBuffer* that indicates the type of data collection hardware installed. The maximum possible value returned is ITC_DEVID_SCANHW_MAX.
  - **ITC_DEVID_SCANHW_NONE**
    No scanner hardware is installed.
  - **ITC_DEVID_OEM2D_IMAGER**
    OEM 2D imager is installed.
  - **ITC_DEVID_INTERMEC2D_IMAGER**
    Intermec 2D imager is installed.
  - **ITC_DEVID_SE900_LASER**
    SE900 laser is installed.
  - **ITC_DEVID_SE900HS_LASER**
    SE900HS laser is installed.

  The high bit indicates whether the S6 scanning engine is installed. The bit mask for this is ITC_DEVID_S6ENGINE_MASK. A non-zero value indicates that the S6 scanning engine is installed.

- **ITC_NVPARM_WAN_INSTALLED**
  This IOCTL reads the state of the WAN radio installed flag. A BOOLEAN DWORD is returned in the buffer pointed to by *lpOutBuffer*. TRUE indicates that the WAN radio is installed. FALSE indicates that no WAN radio is installed.
- **ITC_NVPARM_WAN_FREQUENCY**
  This IOCTL reads the state of the WAN radio frequency flag. A BOOLEAN DWORD is returned in the buffer pointed to by *lpOutBuffer*. TRUE indicates that the WAN radio frequency is United States. FALSE indicates that the WAN radio frequency is European.
- **ITC_NVPARM_WAN_RADIOTYPE**
  This IOCTL reads the WAN radio ID installed by manufacturing. A BYTE is returned in the buffer pointer to by *lpOutBuffer* which indicates the type of WAN radio hardware installed. The maximum possible value returned is ITC_DEVID_WANRADIO_MAX. The current definitions are:
  - **ITC_DEVID_WANRADIO_NONE**
    No WAN radio installed.
  - **ITC_DEVID_WANRADIO_SIERRA_SB555**
    CDMA Sierra Wireless radio.
  - **ITC_DEVID_WANRADIO_XIRCOM_GEM3503**
    GSM/GPRS Intel (Xircom) radio.
  - **ITC_DEVID_WANRADIO_SIEMENS_MC45**
    GSM/GPRS Siemens radio.
- **ITC_NVPARM_80211_INSTALLED**
  This IOCTL reads the state of the 802.11b radio installed flag. A BOOLEAN DWORD is returned in the buffer pointed to by *lpOutBuffer*. TRUE indicates that the 802.11b radio is installed. FALSE indicates that no 802.11b radio is installed.
- **ITC_NVPARM_80211_RADIOTYPE**
  This IOCTL reads the 802.11b radio ID installed by manufacturing. A BYTE is returned in the buffer pointer to by *lpOutBuffer* that indicates the type of 802.11b radio hardware installed. The maximum possible value returned is ITC_DEVID_80211RADIO_MAX. The current definitions are:
  - **ITC_DEVID_80211RADIO_NONE**
    No 802.11b radio installed.
  - **ITC_DEVID_80211RADIO_INTEL_2011B**
    Intel 2011B radio installed.
- **ITC_NVPARM_BLUETOOTH_INSTALLED**
  This IOCTL reads the state of the Bluetooth radio installed flag. A BOOLEAN DWORD is returned in the buffer pointed to by *lpOutBuffer*. TRUE indicates that the Bluetooth radio is installed. FALSE indicates that no Bluetooth radio is installed.

- **ITC_NVPARM_SERIAL2_INSTALLED**
  This IOCTL reads the state of the serial 2 (COM2) device installed flag. A BOOLEAN DWORD is returned in the buffer pointed to by *lpOutBuffer*. TRUE indicates that the serial 2 device is installed. FALSE indicates that no serial 2 device is installed.
- **ITC_NVPARM_VIBRATE_INSTALLED**
  This IOCTL reads the state of the vibrate device installed flag. A BOOLEAN DWORD is returned in the buffer pointed to by *lpOutBuffer*. TRUE indicates that the vibrate device is installed. FALSE indicates that no vibrate device is installed.
- **ITC_NVPARM_LAN9000_INSTALLED**
  This IOCTL reads the state of the Ethernet device installed flag. A BOOLEAN DWORD is returned in the buffer pointed to by *lpOutBuffer*. TRUE indicates that the Ethernet device is installed. FALSE indicates that no Ethernet device is installed.
- **ITC_NVPARM_SIM_PROTECT_HW_INSTALLED**
  This IOCTL reads the state of the SIM card protection hardware installed flag. A BOOLEAN DWORD is returned in the buffer pointed to by *lpOutBuffer*. TRUE indicates that the SIM card protection hardware is installed. FALSE indicates that no SIM card protection hardware is installed.
- **ITC_NVPARM_SIM_PROTECT_SW_INSTALLED**
  This IOCTL reads the state of the SIM card protection software installed flag. A BOOLEAN DWORD is returned in the buffer pointed to by *lpOutBuffer*. TRUE indicates that the SIM card protection software is installed. FALSE indicates that no SIM card protection software is installed.

IOCTL_HAL_ITC_WRITE_SYSPARM

Describes and enables the registry save location.

Usage

#include “oemioctl.h”

Syntax

```
BOOL KernelIoControl( IOCTL_HAL_ITC_WRITE_SYSPARM,LPVOID
lpInBuf,DWORD nInBufSize, LPVOID lpOutBuf, DWORD
nOutBufSize, LPDWORD lpBytesReturned );
```

Parameters

lpInBuf	A single byte that may be one of the *id* values. See “*ID Field Values*” below.
nInBufSize	Must be set to the size of the *lpInBuf* in bytes.
lpOutBuf	Must point to a buffer large enough to hold the data to be written to the non-volatile data store.
nOutBufSize	The size of *lpOutBuf* in bytes.
lpBytesReturned	The number of bytes returned by the function.

Return Values

Returns TRUE if function succeeds. Returns FALSE if the function fails. GetLastError() may be used to get the error value. Either ERROR_INVALID_PARAMETER or ERROR_INSUFFICIENT_BUFFER may be returned when this function is used to get the error.

ID Field Values

The *id* field of *lpInBuf* may be one of the following values:

- **ITC_REGISTRY_LOCATION**
  This IOCTL sets the default location for where to write the registry when RegFlushKey() is called by an application. The registry may be saved to Flash, a CompactFlash storage card or a SecureDigital storage card. *lpOutBuf* must point to a buffer that contains a byte value of “1” for the CompactFlash card or “2” for the SecureDigital card to specify the location.
- **ITC_REGISTRY_SAVE_ENABLE**
  This function enables or disables the save registry to non-volatile media feature of the RegFlushKey() function. *lpOutBuf* must be set to zero (FALSE) if the feature is to be disabled or one (TRUE) if the feature is to be enabled.
- **ITC_ DOCK_SWITCH**
  This IOCTL sets a position of the dock switch. The dock switch may be set to either “modem” or “serial” positions. *lpOutBuf* must point to a buffer that contains a byte value of either DOCK_MODEM or DOCK_SERIAL as defined in OEMIOCTL.H; the value specifies the position the switch is to be set.

- **ITC_ WAKEUP_MASK**
  This IOCTL sets a bit mask that represents the mask for the five programmable wakeup keys. The I/O key is not a programmable wakeup key. By default it is always the system resume key and all other keys are set to disable key wakeup. A zero in a bit position masks the wakeup for that key. A one in a bit position enables wakeup for that key. *lpOutBuf* must point to a buffer that contains a byte value of a wakeup mask consisting of the OR'ed constants as defined in OEMIOCTL.H. Only the following keys are programmable as wakeup events.

```
#define   SCANNER_TRIGGER 1
#define   SCANNER_LEFT    2
#define   SCANNER_RIGHT   4
#define   GOLD_A1         8
#define   GOLD_A2         0x10
```

- **ITC_AMBIENT_KEYBOARD**
  This IOCTL sets the threshold for the keyboard ambient sensor. This can be a value from 0 (always off) to 255 (always on). *lpOutBuf* must point to a buffer that contains a byte value of the desired setting.
- **ITC_AMBIENT_FRONTLIGHT**
  This IOCTL sets the threshold for the frontlight ambient sensor. This can be a value from 0 (always off) to 255. *lpOutBuf* must point to a buffer that contains a byte value of the desired setting.

IOCTL_HAL_GET_DEVICEID

This IOCTL returns the device ID. There are two types of device IDs supported, which are differentiated based on the size of the *output* buffer. The UUID is returned if the buffer size is set to *sizeof(UNIQUE_DEVICEID)*, otherwise the oldstyle device ID is returned.

Usage

#include “pkfuncs.h”
#include “deviceid.h”

Syntax

```
BOOL KernelIoControl( IOCTL_HAL_GET_DEVICEID,LPVOID
lpInBuf,DWORD nInBufSize,LPVOID lpOutBuf,DWORD
nOutBufSize,LPDWORD lpBytesReturned );
```

Parameters

lpInBuf	Should be set to NULL. STRICT_ID settings are not supported.
lpInBufSize	Should be set to zero.
lpOutBuf	Must point to a UNIQUE_DEVICEID structure as defined by DEVICEID.H if the UUID is to be returned.
nOutBufSize	The size of the UNIQUE_DEVICEID in bytes if the UUID is to be returned. A DEVICE_ID as defined by PKFUNCS.H is returned if the size in bytes is greater than or equal to *sizeof(DEVICE_ID)*.
lpBytesReturned	The number of bytes returned by the function.

Return Values

Returns TRUE if function succeeds. Returns FALSE if the function fails. GetLastError() may be used to get the extended error value.

IOCTL_HAL_GET_OAL_VERINFO

Returns the HAL version information of the Pocket PC image.

Usage

#include “oemioctl.h”

Syntax

```
BOOL KernelIoControl( IOCTL_HAL_GET_OAL_VERINFO,LPVOID
lpInBuf,DWORD nInBufSize,LPVOID lpOutBuf,DWORD
nOutBufSize,LPDWORD lpBytesReturned );
```

Parameters

lpInBuf Should be set to NULL.

lpInBufSize Should be set to zero.

lpOutBuf Must point to a VERSIONINFO structure as defined by OEMIOCTL.H. The fields should have these values:

- `cboemverinfo` sizeof (tagOemVerInfo);
- `verinfover` 1
- `sig;` “ITC\0”
- `id;` ‘N’
- `tgtcustomer` “”
- `tgtplat` SeaRay
- `tgtplatversion` Current build version number
- `tgtcputype[8];` “Intel\0”
- `tgtcpu` “PXA250\0”;
- `tgtcoreversion` “”
- `date` Build time
- `time` Build date

nOutBufSize The size of VERSIONINFO in bytes.

lpBytesReturned Returns *sizeof(PVERSIONINFO)*.

Return Values

Returns TRUE if function succeeds. Returns FALSE if the function fails. GetLastError() may be used to get the extended error value.

IOCTL_HAL_GET_BOOTLOADER_VERINFO

Returns the HAL version information of the Pocket PC image.

Usage

#include “oemioctl.h”

Syntax

```
BOOL KernelIoControl( IOCTL_HAL_GET_OAL_VERINFO,LPVOID
lpInBuf, DWORD nInBufSize,LPVOID lpOutBuf,DWORD
nOutBufSize,LPDWORD lpBytesReturned );
```

Parameters

lpInBuf Should be set to NULL.

lpInBufSize Should be set to zero.

lpOutBuf Must point to a VERSIONINFO structure as defined by OEMIOCTL.H. The fields should have these values:

- `cboemverinfo` Sizeof (tagOemVerInfo);
- `verinfover` 1
- `sig;` “ITC\0”
- `id;` ‘B’
- `tgtcustomer` “”
- `tgtplat` SeaRay
- `tgtplatversion` Current build version number of the bootstrap loader
- `tgtcputype[8];` “Intel\0”;
- `tgtcpu` “PXA250\0”
- `tgtcoreversion` “”
- `date` Build time
- `time` Build date

nOutBufSize The size of VERSIONINFO in bytes.

lpBytesReturned The number of bytes returned to *lpOutBuf*.

Return Values

Returns TRUE if function succeeds. Returns FALSE if the function fails. GetLastError() may be used to get the extended error value.

IOCTL_HAL_WARMBOOT

Causes the system to perform a warm-boot. The object store is retained.

Usage
#include “oemioctl.h”

Syntax
```
BOOL KernelIoControl( IOCTL_HAL_WARMBOOT,LPVOID
lpInBuf,DWORD nInBufSize,LPVOID lpOutBuf,DWORD
nOutBufSize,LPDWORD lpBytesReturned );
```

Parameters
lpInBuf Should be set to NULL.

lpInBufSize Should be set to zero.

lpOutBuf Should be NULL.

nOutBufSize Should be zero.

Return Values
None.

IOCTL_HAL_COLDBOOT

Causes the system to perform a cold-boot. The object store is cleared.

Usage
#include “oemioctl.h”

Syntax
```
BOOL KernelIoControl( IOCTL_HAL_COLDBOOT,LPVOID
lpInBuf,DWORD nInBufSize,LPVOID lpOutBuf,DWORD
nOutBufSize,LPDWORD lpBytesReturned );
```

Parameters
lpInBuf Should be set to NULL.

lpInBufSize Should be set to zero.

lpOutBuf Should be NULL.

nOutBufSize Should be zero.

Return Values
None.

IOCTL_HAL_GET_RESET_INFO

This IOCTL code allows software to check the type of the most recent reset.

Usage

#include “oemioctl.h”

Syntax

```
BOOL KernelIoControl( IOCTL_HAL_GET_RESET_INFO,LPVOID
lpInBuf,DWORD nInBufSize,LPVOID lpOutBuf,DWORD
nOutBufSize,LPDWORD lpBytesReturned );
```

Parameters

lpInBuf Should be set to NULL.

lpInBufSize Should be set to zero.

lpOutBuf Must point to a HAL_RESET_INFO structure:

```
typedef struct {
   DWORD ResetReason;                       // most recent reset type
   DWORD ObjectStoreState;                  // state of object store
} HAL_RESET_INFO, * PHAL_RESET_INFO;

// Reset reason types
#define HAL_RESET_TYPE_UNKNOWN          0
#define HAL_RESET_REASON_HARDWARE       1  // cold
#define HAL_RESET_REASON_SOFTWARE       2  // suspend
#define HAL_RESET_REASON_WATCHDOG       4
#define HAL_RESET_BATT_FAULT            8  // power fail
#define HAL_RESET_VDD_FAULT             16 // warm boot

// Object store state flags
#define HAL_OBJECT_STORE_STATE_UNKNOWN  0
#define HAL_OBJECT_STORE_STATE_CLEAR    1
```

nOutBufSize The size of HAL_RESET_INFO in bytes.

lpBytesReturned The number of bytes returned by the function.

Return Values

Returns TRUE if function succeeds. Returns FALSE if the function fails. GetLastError() may be used to get the extended error value.

IOCTL_HAL_GET_BOOT_DEVICE

This IOCTL code allows software to check which device CE booted from.

Usage

#include “oemioctl.h”

Syntax

```
BOOL KernelIoControl( IOCTL_HAL_GET_BOOT_DEVICE,LPVOID
lpInBuf,DWORD nInBufSize,LPVOID lpOutBuf,DWORD
nOutBufSize,LPDWORD lpBytesReturned );
```

Parameters

lpInBuf Should be set to NULL.

lpInBufSize Should be set to zero.

lpOutBuf Must point to a buffer large enough to hold a DWORD (4 bytes) that contains the boot device. The following boot devices are supported:

```
#define HAL_BOOT_DEVICE_UNKNOWN        0
#define HAL_BOOT_DEVICE_ROM_XIP        1
#define HAL_BOOT_DEVICE_ROM            2
#define HAL_BOOT_DEVICE_PCMCIA_ATA     3
#define HAL_BOOT_DEVICE_PCMCIA_LINEAR  4
#define HAL_BOOT_DEVICE_IDE_ATA        5
#define HAL_BOOT_DEVICE_IDE_ATAPI      6
```

nOutBufSize The size of lpOutBuf in bytes (4).

lpBytesReturned The number of bytes returned by the function.

Return Values

Returns TRUE if function succeeds. Returns FALSE if the function fails. GetLastError() may be used to get the extended error value.

IOCTL_HAL_REBOOT

Causes the system to perform a warm-boot. The object store is retained.

Usage

#include “oemioctl.h”

Syntax

```
BOOL KernelIoControl( IOCTL_HAL_REBOOT,LPVOID lpInBuf,DWORD
nInBufSize,LPVOID lpOutBuf,DWORD nOutBufSize,LPDWORD
lpBytesReturned );
```

Parameters

lpInBuf	Should be set to NULL.
lpInBufSize	Should be set to zero.
lpOutBuf	Should be NULL.
nOutBufSize	Should be zero.

Return Values

None.

IOCTL_PROCESSOR_INFORMATION

Returns processor information.

Usage

#include “pkfuncs.h”

Syntax

```
BOOL KernelIoControl( IOCTL_PROCESSOR_INFORMATION,LPVOID
lpInBuf,DWORD nInBufSize,LPVOID lpOutBuf,DWORD
nOutBufSize,LPDWORD lpBytesReturned );
```

Parameters

Parameters:

lpInBuf Should be set to NULL.

lpInBufSize Should be set to zero.

lpOutBuf Should be a pointer to the PROCESSOR_INFO structure. The PROCESSOR_INFO structure stores information that describes the CPU more descriptively.

```
typedef __PROCESSOR_INFO {
WORD    wVersion;              // Set to value 1
WCHAR   szProcessorCore[40];   // “ARM\0”
WORD    wCoreRevision;         // 4
WCHAR   szProcessorName[40];   // “PXA250\0”
WORD    wProcessorRevision;    // 0
WCAHR   szCatalogNumber[100];  // 0
WCHAR   szVendor[100];         // “Intel Corporation\0”
DWORD   dwInstructionSet;      // 0
DWORD   dwClockSpeed;          // 400
}
```

nOutBufSize Should be set to sizeof(PROCESSOR_INFO) in bytes.

lpBytesReturned Returns sizeof(PROCESSOR_INFO);

Return Values

Returns TRUE if function succeeds. Returns FALSE if the function fails. GetLastError() may be used to get the extended error value.

IOCTL_GET_CPU_ID

Returns Xscale processor ID.

Usage
#include “oemioctl.h”

Syntax
```
BOOL KernelIoControl( IOCTL_GET_CPU_ID,LPVOID lpInBuf, DWORD
nInBufSize,LPVOID lpOutBuf,DWORD nOutBufSize,LPDWORD
lpBytesReturned );
```

Parameters

lpInBuf	Should point to a CPUIdInfo structure defined in OEMIOCTL.H.
lpInBufSize	Should be sizeof(CPUIdInfo).
lpOutBuf	Should be NULL.
nOutBufSize	Should be set to 0.
lpBytesReturned	Returns sizeof(PROCESSOR_INFO);

Return Values
Returns TRUE if function succeeds. Returns FALSE if the function fails. GetLastError() may be used to get the extended error value.

Reboot Functions

There are several methods, via Kernel I/O Control functions, that an application program can use to force the 700 Series Computer to reboot.

IOCTL_HAL_REBOOT

IOCTL_HAL_REBOOT performs a warm-boot. See page 278.

IOCTL_HAL_COLDBOOT

Invoking the KernelIOControl function with IOCTL_HAL_COLDBOOT forces a cold reboot. This resets the 700 Series Computer and reloads Windows CE as if a power-up had been performed. The contents of the Windows CE RAM-based object store are discarded. See page 275.

IOCTL_HAL_WARMBOOT

This function is supported on the 700 Series Computers. It performs a warm boot of the system, preserving the object store. See page 275.

Remapping the Keypad

Note; Use caution when remapping the keypad. Improper remapping may render the keypad unusable. Data within the 700 Series Computer could also be lost, should any problems occur.

Applications have the ability to remap keys on the 700 Color Keypad. This will allow applications to enable keys that would otherwise not be available, such as the [F1] function key. Also, to disable keys that should not be available, such as the alpha key because no alpha entry is required. Care should be exercised when attempting to remap the keypad because improper remapping may cause the keypad to become unusable. This can be corrected by cold booting the device which will cause the default keymap to be loaded again.

Note that remapping the keys in this way affects the key mapping for the entire system, not just for the application that does the remapping.

There are three "planes" supported for the 740 Keypad. Keys that are to be used in more than one shift plane must be described in each plane.

Unshifted Plane

The unshifted plane contains values from the keypad when not pressed with other keys, such as the following:

- [**1**] 1
- [5] 5
- [**9**] 9

Gold Plane

The gold plane contains values from the keypad when a key is simultaneously pressed with the [**Gold**] key, such as the following:

- [**Gold**] + [**1**] Send
- [**Gold**] + [5] A3
- [**Gold**] + [**9**] PageDown

Alpha Plane

The alpha plane contains values from the keypad when the keypad has been placed in alpha mode by pressing the blue alpha key, such as the following:

- [**Alpha**] + [**1**] Caps
- [**Alpha**] + [5] JKL
- [**Alpha**] + [**9**] WXYZ

Key Values

Key values for each plane are stored in the registry. All units ship with a default key mapping already loaded in the registry. Applications that wish to change the default mapping need to read the appropriate key from the registry into an array of Words, modify the values required and then write the updated values back into the registry. The registry access can be done with standard Microsoft API calls, such as RegOpenKeyEx(), RegQueryValueEx(), and RegSetValueEx().

- The unshifted plane mapping can be found in the registry at:
  `HKEY_LOCAL_MACHINE\HARDWARE\DEVICEMAP\KEYBD\Vkey`
- The gold plane mapping can be found in the registry at:
  `HKEY_LOCAL_MACHINE\HARDWARE\DEVICEMAP\KEYBD\VkeyGold`
- The alpha plane mapping can be found in the registry at:
  `HKEY_LOCAL_MACHINE\HARDWARE\DEVICEMAP\KEYBD\VkeyAlpha`

How Key Values Are Stored in Registry

To know which fields to update in the registry, you must know what Scan Codes are assigned to each physical key (see the table below). The Scan Code is used at the lowest level of the system to let the keypad driver know which physical key has been pressed. The keypad driver takes that scan code and looks it up in a table (a copy of the one stored in the registry) to determine which values to pass on to the operating system.

Each registry key is just an array that describes to the keypad driver what value needs to be passed for each physical key. The key values are indexed by the scan code, this is a zero-based index. For example in the unshifted plane, the [**4**] key has a scan code of 0x06. This means that the seventh word under the "Vkey" registry key will have the value for the [**4**] key. Taking a sample of the "Vkey" registry key shows the following values:

```
00,00,0B,05,02,03,C1,07,04,03,BE,00,34,00,00,00,. . .
```

The value is 34,00. The values are in reverse byte order because that is the way the processor handles data. When writing an application, nothing needs to be done to swap the bytes, as this will happen automatically when the data is read into a byte value. This is something you just need to be aware of this when looking at the registry. Knowing this, we can see that the value that the keypad driver will pass to the system is a hex 34. Looking that up on an UNICODE character chart, we see that it maps to a "4". If you wanted the key, labeled "4", to output the letter "A" instead, you would need to change the seventh word to "41" (the hexadecimal representation of "A" from the UNICODE chart), then put the key back into the registry.

Note: Do not remap scan codes 0x01, 0x41, 0x42, 0x43, 0x44. Remapping these scan codes could render your 700 Series Computer unusable until a cold-boot is performed.

If you wish to disable a certain key, remap its scan code to 0x00.

Change Notification

Just changing the registry keys will not immediately change the key mappings. To notify the keypad driver that the registry has been updated, signal the "ITC_KEYBOARD_CHANGE" named event using the CreateEvent() API.

Advanced Keypad Remapping

It is also possible to map multiple key presses to one button and to map named system events to a button. The multiple key press option could be useful to cut down on the number of keys needed to press in a given situation or to remap which key behaves like the action key. Mapping events to a button could be useful to change which buttons will fire the scanner, control volume, and allow for suspending and resuming the device. If you need help performing one of these advanced topics please contact Intermec Technical Support.

Scan Codes

At the lowest driver level, the 740 Keypad identifies keys as scan codes. These scan codes are sent via the keypad microcontroller, and cannot be changed without modifying the keypad firmware.

Key/Meaning	Scancode
Reserved	0x00
I/O Button	0x01
Scanner Trigger	0x02
Scanner Left	0x03
Scanner Right	0x04
.	0x05
4	0x06
None	0x07
Left Arrow	0x08
None	0x09
Backspace	0x0A
Gold Key	0x0B
None	0x0C
ESC	0x0D
Down Arrow	0x0E
1	0x0F
7	0x10
Alpha Key	0x11
None	0x12
Up Arrow	0x13
Right Arrow	0x14
2	0x15
8	0x16
0	0x17
5	0x18
None	0x19

Key/Meaning	Scancode
Action Key	0x1A
3	0x1B
9	0x1C
ENTER	0x1D
6	0x1E
None	0x1F-0x40
Charge Detect	0x41
LCD Frontlight	0x42
Ambient Light	0x42
Threshold Crossed	0x42
Headset Detected	0x43
Keypad Backlight	0x44
Ambient Light	0x44
Threshold Crossed	0x44

Sample View of Registry Keys

The following is a sample view of the current default key mapping. See the registry on your device for the latest key mappings.

```
[HKEY_LOCAL_MACHINE\HARDWARE\DEVICEMAP\KEYBD]
"ResumeMask"=dword:7
"Vkey"=hex: 00,00,0B,05,02,03,C1,07,04,03,BE,00,34,00,00,00,\
  25,00,00,00,08,00,03,02,00,00,1B,00,28,00,31,00,\
  37,00,01,02,00,00,26,00,27,00,32,00,38,00,30,00,\
  35,00,00,00,01,03,33,00,39,00,0D,00,36,00,00,00,\
  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,\
  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,\
  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,\
  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,\
  00,00,07,05,01,05,03,05,02,05

"VkeyGold"=hex:  00,00,0B,05,02,03,C1,07,04,03,BE,00,34,00,00,00,\
  09,01,00,00,BF,00,03,02,00,00,BD,00,75,00,72,00,\
  21,00,01,02,00,00,76,00,09,00,73,00,38,01,5B,00,\
  35,00,00,00,BB,01,09,05,22,00,32,01,36,00,00,00,\
  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,\
  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,\
  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,\
  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,\
  00,00,07,05,01,05,03,05,02,05

"VkeyAlpha"=hex: 00,00,0B,05,02,03,C1,07,04,03,BE,00,47,00,00,00,\
  25,00,00,00,08,00,03,02,00,00,1B,00,28,00,02,02,\
  50,00,01,02,00,00,26,00,27,00,41,00,54,00,20,00,\
  4A,00,00,00,01,03,44,00,57,00,0D,00,4D,00,00,00,\
  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,\
  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,\
  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,\
  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,\
  00,00,07,05,01,05,03,05,02,05
```