Programmer's Reference Guide

Windows XPS Driver Software Development Kit

October 2020

527250-003, Rev. B

Notice

Please do not attempt to operate or repair this equipment without adequate training. Any use, operation or repair you perform that is not in accordance with the information contained in this documentation is at your own risk.

Trademark Acknowledgments

Entrust, Sigma and the hexagon design are trademarks, registered trademarks and/or service marks of the Entrust Corporation in the United States and other countries.

Datacard is a registered trademark and service mark of Entrust Corporation in the United States and other countries.

MasterCard is a registered trademark of MasterCard International Incorporated.

Visa is a registered trademark of Visa International Service Association.

All other product names are the property of their respective owners.

Proprietary Notice

The design and information contained in these materials are protected by US and international copyright laws.

All drawings and information herein are the property of Entrust Corporation. All unauthorized use and reproduction is prohibited.

Entrust Corporation
1187 Park Place
Shakopee, MN 55379
Phone: 952-933-1223
Fax: 952-933-7971
www.entrust.com

Safety

The following basic safety tips are given to ensure safe installation, operation and maintenance of Datacard® equipment.

- Connect equipment to a grounded power source. Do not defeat or bypass the ground lead.
- Place the equipment on a stable surface (table) and ensure floors in the work area are dry and non-slip.
- Know the location of equipment branch circuit interrupters or circuit breakers and how to turn them on and off in case of emergency.
- Know the location of fire extinguishers and how to use them. ABC type extinguishers may be used on electrical fires.
- Know local procedures for first aid and emergency assistance at the customer facility.
- Use adequate lighting at the equipment location.
- Maintain the recommended temperature and humidity range in the equipment area.

Safe Human Interface

- Use proper lifting techniques when moving or installing the equipment.
- Use standard electrostatic discharge (ESD) precautions when working on or near electrical circuits.
- Do not defeat or disconnect safety interlocks on covers.

Warning: To avoid a possible electric shock, always unplug the system before servicing.

Liability

The WARNING and CAUTION labels have been placed on the equipment for your safety. Please do not attempt to operate or repair this equipment without adequate training. Any use, operation, or repair in contravention of this document is at your own risk.

California Prop

Warning: This product contains chemicals, including lead, known to the State of California to cause cancer, and birth defects or other reproductive harm. *Wash hands after handling.*

For more information on this warning, refer to:

www.datacard.com/califpropwarning.

Revision Log

Revision	Date	Description of Changes
A	October 2018	First release of this document with part number 527250-003. This release combines part numbers 527250-001 and 527250-002. The information contained in this document supports the XPS Card Printer Driver version 7.4.
B	October 2020	Updated to support XPS Card Printer Driver version 8.0.

Contents

Chapter 1: Introduction

1

Overview

The Application Programming Interface (API) built into the XPS Card Printer Windows driver (referred to as “the driver” in the remainder of this Guide) provides two methods that your application can use to control card personalization operations through the driver. Both use built-in Windows operating system interfaces.

- **Use the driver Print Ticket.** Print Ticket is a required feature of any driver using the XML Paper Specification (XPS) print driver architecture. A Print Ticket tells the printer how to process a print job. Through Print Ticket, your application can override the driver’s printing preferences on a job-by-job basis.
- **Use the Windows IBidiSpl interface.** The IBidiSpl interface is the Microsoft preferred API for printer control. Using the IBidiSpl interface, your application places the driver in “interactive mode,” where the application has fine-grained job control and can access data on the card during the card personalization process.

  Java does not directly support the IBidiSpl interface. Entrust has created a C++ helper DLL (dxp01sdk_IBidiSpl_interop.dll) that your Java application uses as the interface for interactive printer control. The helper DLL is included with the Software Development Kit (SDK).

The XPS Card Printer Windows Driver SDK (referred to as “the SDK”) includes documentation and sample code that describe and demonstrate how to use both Print Ticket and the IBidiSpl interface.

To learn more about Print Ticket and the IBidiSpl interface, refer to Appendix G: "References”.

The interfaces documented in the SDK provide the following capabilities to your application using the driver.

Not all of the following capabilities of the SDK are available through a Java application. Refer to Chapter 2: "SDK Sample Code" for more information.

- Print while modifying printing characteristics using the Print Ticket:
  - Print one- or two-sided
  - Disable printing on one or both card sides
  - Specify the copy count
  - Print in portrait or landscape orientation
  - Rotate a card side by 180 degrees
  - Select from the predefined topcoat and print blocking
  - Specify the input hopper, including the exception hopper, to use when selecting a card
- Use escaped text in the card data to do the following:
  - Set topcoat blocking rectangles, and set print blocking rectangles
  - Encode standard format magnetic stripe data
  - Specify the input hopper used to select the card
  - Emboss, indent, and top a card when printing to a CE-series or DS4-ES1 printer
- Disable the application of topping foil on embossed cards
- Read magnetic stripe data
- Encode custom magnetic stripe data
- Place an impression on the front or back of a card with the Tactile Impression Module
- Stage a smart card so it can be personalized
- Stage and personalize a smart card using the single-wire smart card interface
- Read and write data to a MIFARE Classic chip smart card
- Specify the side of the card that should face up when placed in the output hopper
- Laminate, debow, and impress a card
- Process more than one job at a time
- Read the bar code on a serialized overlay
- Place pre-serialized cards in a bar code reader so that the bar code can be read

- Send laser engrave data to a laser system
- Check whether the driver or printer is busy and wait before starting a job
- Check the hopper status
- Monitor supplies and printer status
- Get printer and driver error messages
- Recover from printer and driver errors
- Get job status for the current interactive mode job
- Check printer supplies status before printing the card
- Get a count of cards processed by the printer
- Reset the resettable card count values stored in the printer
- Activate or deactivate a printer
- Change the printer state to online, offline, or suspended
- Start the printer cleaning process
- Shut down the printer
- Restart the printer

The SDK supports the same Microsoft Windows operating systems as the driver.

Installation

To use the SDK, extract the XPS Driver SDK zip file to a folder on your computer. For most situations, there are no SDK additional components to install with your application.

The SDK works with the following:

- XPS Card Printer Driver, version 8.0 or newer
- Datacard® SD Series: SD160™, SD260™, SD260L™, SD360™, SD460™ Card Printers
- Datacard® CD Series: CD800™, CD800 with CLM laminator, CD820™ Card Printers
- Datacard® CE840™ Instant Issuance System
- Datacard® CE870™ Kiosk System
- Entrust Datacard™ CR805™ Retransfer Card Printer

- Entrust Datacard™ CL900™ Desktop Laser Personalization System
- Entrust™ Sigma DS2, DS2, and DS3 Direct to Card Printers
- Entrust™ Sigma DS4 Instant Issuance System
- Instant Issuance Systems that are Driver Enabled

A C++ helper DLL (dxp01sdk_IBidiSpl_interop.dll) is included for Java applications because Java cannot interface directly to the IBidiSpl COM interface.

Chapter 2: SDK Sample Code

2

The SDK includes sample code that demonstrates the details you need to successfully use the driver API in your application.

Sample Code

The SDK sample code demonstrates specific card personalization tasks using best practices for Print Ticket usage, job sequencing, and basic error handling. All the samples are console applications to make it easier to integrate the code into your application. Samples are provided in C++, C#, VB.NET, and Java. The C++, C#, and VB.NET samples use direct calls to the IBidiSpl interface. The Java samples use calls to the helper dll (dxp01sdk_IBidiSpl_interop.dll).

- If you use the SDK sample code to send a job to a shared printer, you must specify the fully qualified printer name. Refer to the *XPS Card Printer Driver User's Guide* for complete information about setting up shared printers.
- Several samples include an option to display the Print Ticket data. The default is to not display the data. Java does not have access to the Print Ticket so Java samples cannot display Print Ticket data.

Samples Included in the SDK

The SDK includes the following samples:

	Code Sample			
Sample Function	**cpp**	**csharp**	**vb**	**java**
Bar Code	BarcodePark	barcodePark	BarcodePark	barcode_park
Emboss and Indent	emboss_indent	emboss_indent	emboss_indent	emboss_indent
Lamination	lamination	lamination	lamination	Not available
Laminator Serialized Overlay	lamination_ barcode_read	lamination_ barcode_read	lamination_ barcode_read	Not available
Laser Engraving	laser	laser	laser	laser
Print Locking	locks	locks	locks	locks

Sample Function	Code Sample			
	cpp	**csharp**	**vb**	**java**
Magnetic Stripe	magstripe	magstripe	magstripe	magstripe
Print	print	print	print	print
Printer Control	printer_control	printer_control	printer_control	printer_control
Printer State	printer_state	printer_state	printer_state	printer_state
Smart Card	smartcard	smartcard	smartcard	smartcard
Single-Wire Smart Card	smartcard_ singlewire	smartcard_ singlewire	smartcard_ singlewire	smartcard_ singlewire
Single-Wire Mifare Duali Smart Card	smartcard_ singlewire_duali_ mifare	smartcard_ singlewire_ duali_mifare	smartcard_ singlewire_duali_ mifare	Not available
Single-Wire Omnikey Smart Card	smartcard_ singlewire_ omnikey	smartcard_ singlewire_ omnikey	smartcard_ singlewire_ omnikey	Not available
Status	status	status	status	printer_status

Print Sample (Not Interactive)

Use the Print sample to demonstrate the print functionality of the printer and driver.

The Print sample uses the Print Ticket to override the driver preferences for the following:

- One- or two-sided printing
- Copy count

The copy count option is not used whenever the StartJob command is issued.

- Per card-side portrait or landscape orientation (Java is limited to card-level orientation where both sides of the card have the same orientation)
- Predefined topcoat and print blocking patterns
- Per card-side 180-degree rotation [1]
- Per card-side disabling of printing [1]

[1] Java does not support these features.

The Print sample also demonstrates:

- Color graphics printing
- K (black) text and K graphics printing
- Custom topcoat and print blocking using escapes
- Standard IAT-format magnetic stripe encoding using escapes
- Ability to select the input hopper to use (the default is hopper 1)
- Ability to check the input hopper status before sending the card job
- Ability to poll for job status and error conditions
- Ability to specify the card side (front or back) that faces up when it is placed in the output hopper
- Ability to impress the front or back of a card if the Tactile Impression Module is installed on the printer

Magnetic Stripe Sample

The Magnetic Stripe sample demonstrates ISO or JIS magnetic stripe encoding, with options to read the magnetic stripe data, print text on one or both sides of the card, specify the input hopper from which to select a card, specify the card side on output, and poll for job completion status and error conditions. The print and magnetic stripe data is part of the sample and cannot be changed.

Bar Code Park Sample

The printer must be equipped with the optional bar code reader for this sample to function.

The Barcode Park sample demonstrates parking a card with a pre-serialized bar code in the bar code reader so that the bar code can be read and then moving the card out of the reader. The sample also includes options to park the card so that a bar code on the back of the card can be read (the default is to read a bar code on the front of the card), print text on the card, specify the input hopper from which to select a card, specify the card side on output, and poll for job completion status and error conditions. The print data is part of the sample and cannot be changed. You then can specify whether to continue (the bar code read was successful) or reject (the bar code read failed) the card. The sample prints text if the bar code read is flagged as successful.

Smart Card Sample

The Smart Card sample demonstrates parking a card in the printer smart card reader, moving the card from the reader, and includes options to specify whether the smart card chip is on the back of the card, print on the front of the card, specify the input hopper from which to select a card, specify the card side on output, and poll for job completion status and error conditions. The print data is part of the sample and cannot be changed.

Single-Wire Smart Card Sample

The printer must be equipped with the single-wire smart card option for this sample to function correctly.

The Single-Wire Smart Card sample uses the integrated smart card reader that communicates with the personalization application using the same cable the driver uses to communicate with the printer. It demonstrates parking a card in the printer smart card reader, moving the card from the reader, and includes options to specify whether the smart card chip is on the back of the card, print on the front of the card, specify the input hopper from which to select a card, specify the card side on output, and to poll for job completion status and error conditions. The print data is part of the sample and cannot be changed.

Single-Wire MIFARE Duali Classic Smart Card Sample

The printer must be equipped with the single-wire smart card option. You must use the proper smart cards for this sample to function correctly.

This sample demonstrates smart card operations including reading and writing to the chip for a MIFARE Classic smart card. It uses the single-wire smart card tunnel and Duali reader commands for a MIFARE Classic application. The sample moves the card into and out of the smart card reader, and includes options to specify whether the smart card chip is on the back of the card, print on the front of the card, and to poll for job completion status and error conditions. The print data is part of the sample and cannot be changed.

This sample is not available in Java.

Single-Wire Omnikey Smart Card Sample

The printer must be equipped with the single-wire smart card option for this sample to function correctly.

The Single-Wire Omnikey smart card sample demonstrates parking a card in the printer Omnikey smart card reader, performing single-wire smart card functions, and moving the card from the reader. The Omnikey reader supports multiple cards types, allowing you to select the type of card used from the following: Mifare, iClass, or HIDProx. For each card type, the Omnikey reader performs the following functions:

- Mifare. Reads the card ID and status, and reads and writes data to block 2.
- iClass. Reads the card ID and status, and reads and writes data to block 18.
- HIDProx. Reads the reads the card ID (UID) and facility code (FAC).

The sample includes options to print and poll for job completion. The print data is part of the sample and cannot be changed.

This sample is not available in Java.

Lamination Sample

The printer must be equipped with a laminator for this sample to function.

The Lamination sample demonstrates using Print Ticket to set the lamination options for one or both lamination stations. It overrides the driver printing preferences settings for those options. The sample allows you to specify the laminator to use (L1 or L2), and the sides of the card to laminate. It also includes options to specify the input hopper from which to select a card, specify the card side on output, and to poll for job completion status and error conditions.

This sample is not available in Java.

Read and Verify Laminator Serialized Overlay Sample

The printer must be equipped with a laminator, a bar code scanner, and serialized overlay loaded in the L1 laminator cartridge for this sample to function.

This sample demonstrates using the SDK API to retrieve the value of a serialized overlay bar code from the laminator. It uses the lamination settings specified in the driver, prints a card, and polls for job completion status and error conditions. It includes a verify option, which allows the application to control whether the card should continue or be rejected, based on the value returned. The sample also includes options to specify a wait time to read the bar code data and to save the bar code read data to a file.

This sample is not available in Java.

Emboss and Indent Sample

The Emboss and Indent sample demonstrates the use of escapes to emboss, indent, and apply topping foil to a card using a CE-series or DS4-ES1 instant issuance system. The emboss and indent data is part of the sample and cannot be changed. The sample also includes options to specify an input hopper, disable topping foil application, and poll for job completion status and error conditions.

The Java sample program does not include the following options:

- Disable topping foil application. If you want to disable topping foil, select **Printing preferences** in the Card Printer Driver and set **Layout > Advanced > Embosser topping** to **Off**.
- Display the print ticket data.

Laser Sample

This sample works only with a CL900 desktop laser system.

The Laser sample demonstrates laser engraving on a card in the laser system. The sample specifies the laser setup file to use, and engraves the data on the card. Additional options allow you to transfer laser setup .zip files between the laser system and the computer, and query laser card setups and their elements. The sample also includes options to encode ISO magnetic stripe data, print text on the card (depending on the printer capabilities), specify the input hopper from which to select a card, and poll for job completion status and error conditions. The laser engraving, print, and magnetic stripe data are part of the sample and cannot be changed.

Print Locking Sample

The Print Locking sample demonstrates locking and unlocking the printer using a password (for printers that are equipped with a lock). The sample also allows you to activate or deactivate a printer using the password, to change the password, or set it to a blank password. The sample locks the printer when the password is changed.

Printer Control Sample

The Printer Control sample demonstrates a way to cancel all jobs in the printer, reset cards counts that are resettable, shut down and restart the printer, and start the printer cleaning process. Using this sample to cancel jobs allows you to return the printer to a known good state. In addition to canceling jobs active, or queued, in the printer, any job in an error state in the driver also is canceled.

When you use the printer control sample to start the printer cleaning process, the process completes properly and message, 172: Insert cleaning card is removed from the computer. However, the job is not removed from the print queue. If there is more than one cleaning card job in the print queue, the insert cleaning card message continues to display on the computer monitor.

The Printer Control sample also demonstrates changing the red color settings and setting the default red and green color settings for SD-, CD-, and CE-series direct-to-card printers running D3.17.4 or newer firmware, and Sigma direct-to-card printers running D4 firmware.

Printer State Sample

The Printer State sample demonstrates changing the printer state to offline, online, or suspended.

Status Sample

The Status sample demonstrates using interactive mode to retrieve printer options, printer status messages, card counts, supply information, job completion status, and error conditions. The sample also allows you to start a job and includes options to specify the input hopper from which to select a card and specify the card side on output.

Sample Code Location

The sample source code is located in the **samples** folder. Select the folder that matches the programming language you are interested in, and then select the folder for the sample containing the features you want to learn about.

Compiled versions of the samples for Visual C++, Visual C#, and VB.NET are included in the **exes** folder. These allow you to demonstrate the sample code without having to build the code yourself. Each sample includes help text that describes the parameters you can enter. To view the help from a command line, open the appropriate language folder, and select the operating system folder for your computer. Open a command prompt window and enter the full path name of the sample with no parameters to display help information and command line options.

The compiled samples have the following runtime dependencies.

- **C++:** Requires Microsoft Visual C++ 2015 Redistributable Package (x86 and x64), or newer. Use the following link to download the appropriate software package:

  https://www.microsoft.com/en-us/download/details.aspx?id=48145

- **C# and VB.NET:** Require Microsoft .NET v4 Client Framework.

The Java sample programs included in the **samples** folder require that you first create a common_java.jar file and then create a runnable JAR file for each sample that you want to use. Refer to Appendix B: "Use Eclipse to Create Java Samples" for complete information.

In addition, runnable Java JAR files for the sample programs are included in the **jars** folder. You can run these JAR files without having to build them. Refer to Appendix C: "Use the SDK Java Samples" for complete information.

Developer Environments

The sample code was developed using the following tools. You are not required to use these, but their use guarantees that the sample code builds without issue.

- **C++, C#, and VB.NET:** Microsoft Visual Studio 2015, or newer. (You can use any edition, including the free Express Edition, for C# and VB.NET. Visual C++ requires the Professional edition at a minimum.)

  You can update the Visual Studio project to use Visual Studio 2017 or Visual Studio 2019. You need to disable Spectre mitigation if your PC does not have runtime libraries with mitigation enabled. The /Qspectre option is available in Visual Studio 2017 version 15.5. 5 and later, and in Visual Studio 2015 Update 3 through KB 4338871.
  More information is available at:
  https://docs.microsoft.com/en-us/cpp/build/reference/qspectre?view=vs-2019

- **Java:** Eclipse Oxygen release. Appendix B: "Use Eclipse to Create Java Samples" contains step-by-step instructions for building the SDK Java sample code with Eclipse. In addition the Java helper DLL requires that the Microsoft Visual C++ 2013, or newer, Redistributable Package be installed. The download link is shown in "Sample Code Location".

  The runnable JAR files included with the SDK in the jars folder were built and tested using Java 8 update 181. We recommend you use this Java version with the samples. Refer to Appendix C: "Use the SDK Java Samples" for more information.

Printing

Your application can either print or block areas on the card from printing using conventional printing APIs along with escapes. This method is always used, even when a job includes interactive mode operations for other card personalization tasks or monitoring job status.

Using Print Ticket, a Microsoft Visual C++, C#, or VB.NET application can override any of the printing preferences set in the driver's Printing Preferences window.

Java printing does not have access to the Print Ticket, so Java applications do not support the following options:

- Rotate the front side or back side image
- Disable printing

In addition, Java applications limit the card orientation (portrait or landscape) to the entire card, not per side.

The driver separates the print items into separate images expected by the printer (color, monochrome, UV, luster). The images that are created are based on both of the following:

- The type of print items in the card design
- The type of ribbon installed in the printer

The Card Printer Driver always uses the ribbon panels designated for the current card side and adjusts the print items on the card to create the best possible image using those panels.

The following sections describe rules for rendering card design elements.

Text Printing

The driver uses the following rules to determine which panels are used to print text:

- If the printer has a color ribbon, any text that is 100% opaque and pure black is rendered by the monochrome black (K) ribbon panel if one is available for the current card side. Otherwise, black text is rendered using the color panels. Text that is 100% opaque and pure white is “punched out” of both the color and monochrome panels. In other words, the white text is created by not printing any color so the white card background shows through. All other text is rendered using the color ribbon panels.
- If the printer has a monochrome ribbon, all non-white text is converted to pure black and prints the same as pure black text would. Pure white text is punched out of any color surrounding it.
- If the printer has a ribbon that includes an ultraviolet (F) or luster (L) panel, text that is 100% opaque and is set at RGB(217,217,217) is rendered by the appropriate panel.

Raster Graphics Printing

Raster graphics are images with formats such as bmp, jpeg, png, and tiff.

The driver uses the following rules to determine which panels to use when printing a raster graphic:

- If the printer has a color ribbon with a K panel on the current card side, a raster graphic is rendered by the monochrome (K) ribbon panel when any of the following are true:
  - It is a 2-color (1 bpp) image with black being one of the colors

    OR
  - It is a 100% opaque image with only pure black and pure white pixels

    OR
  - An image contains any black pixels and the printing preference “Print black image pixels using monochrome” is enabled. In this case, only the near-black pixels are printed with the K panel.

  All other images are rendered to the color panels.

  Due to the way JPEG compresses images, it is unlikely that a JPEG image will ever have only black and white pixels.

- If the printer has a monochrome ribbon, all raster graphics are rendered by the monochrome (K) ribbon panel. Images that normally would be rendered to the color panels (for example, photos) are half-toned to preserve the image details.
- If the printer has a ribbon with a UV (F) or luster (L) panel, a raster graphic is rendered by the F or L panel when it is a 100% opaque image where one color is RGB(217,217,217) and the other color is pure white.

Vector Graphics Printing

Vector graphics are images with formats, such as WMF. These images are represented by a series of commands that draw graphic objects to create the complete image. Most vector graphics elements have an outside border (the stroke) and an inside color (the fill).

The driver uses the following rules to determine which panels are used to print a vector graphic element.

- If the printer has a color ribbon with a K panel on the current card side, a vector graphic is rendered by the monochrome (K) ribbon panel when:
  - There is no fill and the stroke is 100% opaque and pure black

    OR
  - There is no stroke and the fill is 100% opaque and pure black

    OR
  - Both the fill and stroke are 100% opaque and pure black

  All other elements are rendered to the color panels.
- If the printer has a monochrome ribbon, all vector graphic elements are rendered by the monochrome (K) ribbon panel. Elements that would normally be rendered to the color panels are half-toned to make them appear as a shade of gray.
- If the printer ribbon includes a UV (F) or luster (L) panel, a vector graphic element is rendered by the F or L panel when it is 100% opaque and is set to RGB(217,217,217).

Topcoat and Print Blocking

A card design may have features that must not be printed on or have topcoat applied over them. Examples include a smart card chip, a magnetic stripe, and a signature panel. Using escapes, you can specify rectangles to block printing, block topcoat, or apply topcoat. Details about using escapes for blocking printing and topcoat can be found in the “Print Blocking Escapes” section of the *XPS Card Printer Driver User’s Guide*.

A retransfer card printer blocks printing on the back side of the card only if the ink ribbon includes an inhibitor panel on the back side panel set. When you use escapes to specify a non-printing area over a smart card chip on the front of the card, the primer panel is not applied, preventing the retransfer film (and any printing) from adhering to the card. Refer to the “Print Blocking in a Retransfer Printer” section of the *XPS Card Printer Driver User’s Guide* for complete information.

For more information on non-printing areas, refer to the “Non-Printing Areas” section of your printer’s *Installation and Administrator’s Guide*.

Controlling Card Printing Preferences

The Windows printing interface allows job-level application control of the following:

- Card orientation (portrait or landscape)
- Two-sided printing
- Copy count

Applications written in Microsoft Visual C++, C#, and VB.NET can use the Print Ticket to access custom preferences created just for the XPS Card Printer Driver. The custom preferences are:

- Per side card orientation
- Per side 180-degree card image rotation
- Per side disable printing flag that ignores the print data in the job
- Selection of one of the print and topcoat blocking preset masks
- Input hopper used to select the card
- Split-ribbon color printing (non-retransfer printers only)
- Lamination, debow, and impress actions
- A second layer of retransfer material applied to the card (retransfer card printers only)

Sample Code that Demonstrates Printing

The SDK includes sample code with language-specific implementation details for printing. The samples are:

Language	Sample Code	Compiled Samples
Visual C++, Visual C#, and VB.NET	print	exes
Java	print	Refer to Appendix B: "Use Eclipse to Create Java Samples" for information about how to create a runnable JAR file. Refer to Appendix C: "Use the SDK Java Samples" for information about how to use the runnable JAR files shipped with the SDK.

Get the Status of a Print Job

Your application can retrieve the status for the current print job to determine whether the printer is still actively processing the card.

PrinterJobID is used to identify the job. The printer job ID is retrieved by calling `Printer.PrintMessages:Read` after the print job has been submitted to the printer. Once the printer job ID is known, the job status can be retrieved using `Printer.JobStatus:Read` with the PrinterJobID of the current job. Refer to "Get the Status of an Interactive Job" on page 29.

Embossing

Your application can use escapes to emboss, indent, and apply topping foil to a card sent to a CE-series or DS4-ES1 instant issuance system. The Visual C++, C#, and VB.NET emboss_indent samples also allow you to disable the application of topping foil.

The Java sample does not support disabling the topping foil (refer to "Emboss and Indent Sample" on page 10).

Escapes that control embossing and indenting are designed to work across a wide range of applications. The escapes rely on special text character sequences to alert the driver that the text that follows is meant as a command and is not to be printed.

For more information about embosser escapes, including examples and limitations, refer to the *XPS Card Printer Driver User's Guide*.

Sample Code that Demonstrates Embossing

For working code that demonstrates embossing, indenting, and topping, refer to the following samples:

Language	Sample Code	Compiled Samples
Visual C++, Visual C#, and VB.NET	emboss_indent	exes
Java	emboss_indent	Refer to Appendix B: "Use Eclipse to Create Java Samples" for information about how to create a runnable JAR file. Refer to Appendix C: "Use the SDK Java Samples" for information about how to use the runnable JAR files shipped with the SDK.

Laminating

Your application can laminate and impress a card when using a printer with a laminator. If you plan to use the same lamination settings for all cards, you simply can set the driver's printing preferences. However, your application can override the driver preferences for laminating, either by modifying the job's Print Ticket or by including escapes in the text data for the job.

The SDK sample code demonstrates how to control these operations using the Print Ticket.

Escapes that control lamination and impressing are designed to work across the widest range of applications. The escapes rely on special text character sequences to alert the driver that the text that follows is meant as a command and is not to be printed. For more information about lamination escapes, including examples and limitations, refer to the *XPS Card Printer Driver User's Guide*.

Laminator Bar Code Read

If your system includes a CLM laminator that is equipped with a bar code scanner and you have the proper supplies installed, you can retrieve the unique value printed on each serialized overlay patch by reading the matching bar code printed on the lamination material next to the patch. This value provides your application with a traceable identifier that links the patch applied to the card to the other data used to personalize the card. Reading the bar code is an interactive mode operation. Refer to "Read Data from a Serialized Laminate Bar Code" on page 39.

Sample Code that Demonstrates Laminating

For working code that demonstrates Print Ticket control of laminating, printing, and polling for job status and error conditions, and bar code read, refer to the following samples:

Language	Sample Code	Compiled Samples
Visual C++, Visual C#, and VB.NET	lamination lamination_barcode_read	exes
Java	Java does not include a lamination sample.	

Laser Engraving

Your application can send data to the CL900 laser system that can be laser engraved on a card, query laser elements from a setup file, and import or export card setup files to the laser system. The laser system requires additional laser setup files and card design information that is present on the laser system.

The laser sample uses a byte interface to communicate with the printer driver. Laser engraved images must be base64 binary-encoded. The driver expects the laser data to be in bytes so that it is consistent with the laser interface.

In addition to laser engraving, the laser sample also allows you to:

- Transfer laser setup files between the PC and the laser system. You can specify to import (transfer files from the PC to the laser system) or export (transfer files from the laser system to the PC) the files.
- Retrieve the names of all laser card setup files present in the laser system.
- Retrieve variable laser elements in each laser card setup file.

Refer to the documentation for the laser system for complete information about defining laser setup and card design files.

Sample Code that Demonstrates Laser Engraving

For working code that demonstrates control of laser engraving, magnetic stripe encoding, printing, and polling for job status and error conditions, refer to the following sample:

Language	Sample Code	Compiled Samples
Visual C++, Visual C#, and VB.NET	laser	exes
Java	laser	Refer to Appendix B: "Use Eclipse to Create Java Samples" for information about how to create a runnable JAR file. Refer to Appendix C: "Use the SDK Java Samples" for information about how to use the runnable JAR files shipped with the SDK.

Chapter 3: Interactive Mode Using the IBidiSpl Interface

3

Interactive mode is used when your application needs to control the movement of the card in the printer, retrieve data from the card, or retrieve error and job status information.

Overview

The XPS Card Printer Windows driver uses the Microsoft IBidiSpl interface for bidirectional communication between your application and the printer in interactive mode. The following interactive mode functions are supported by this release of the driver SDK:

- Job control of interactive card personalization functions
- Job control for error detection and recovery
- Encode magnetic stripe
- Read magnetic stripe
- Smart card park (front or back of card)
- Get supplies and printer status
- Single-wire smart card park and personalization
- Read serialized patch overlay bar code
- Place pre-serialized cards in a bar code reader so that the bar code can be read
- Send laser engrave data to a laser system
- Monitor and reset card counts
- Get installed printer options
- Get the input hopper status (retransfer card printers only)
- Specify the input hopper (1–6) to use when selecting a card
- Specify the side of the card (front or back) that faces up when it is placed in the output hopper
- Lock and unlock a printer with locks, and change a password
- Activate or deactivate the printer

- Change the printer state
- Change color settings (SD-, CD-, CE-series direct-to-card printers running D3.17.4 or newer firmware, and Sigma direct-to-card printers running D4 firmware)
- Shut down and restart the printer
- Start the printer cleaning cycle
- Impress the front or back of the card if the Tactile Impression Module is installed

Printing, magnetic stripe encoding using escapes or fonts, topcoating, embossing and indenting, laminating, and impressing are done outside interactive mode, but can be mixed with interactive functions within the same job.

Java does not have direct access to the IBidiSpl interface. A C++ helper DLL (dxp01sdk_IBidiSpl_interop.dll) is provided with the SDK that Java applications use for interactive mode.

Interactive Operations

The following IBidiSpl requests and Java helper DLL functions are used to implement the functions described in the "Overview" on page 21:

IBidiSpl Requests	Java Helper DLL Interface Functions
Job Control (normal)	
• Printer.Print:StartJob:Set • Printer.Print.EndJob:Set • Printer.Action:Set • Printer.JobStatus:Read	• StartJob2 • EndJob • ResumeJob • GetJobStatusXML
Job control (error state)	
• Printer.PrintMessages:Read • Printer.Action:Set	• PrinterStatusXML.GetPrinterMessages • CancelJob

IBidiSpl Requests	Java Helper DLL Interface Functions
Card personalization	
• Printer.MagstripeUnit:Back:Encode • Printer.MagstripeUnit:Back:Read • Printer.MagstripeUnit:Front:Encode • Printer.MagstripeUnit:Front:Read • Printer.BarcodeUnit:Front:Park • Printer.BarcodeUnit:Back:Park • Printer.SmartCardUnit:Front:Park • Printer.SmartCardUnit:Back:Park • Printer.SmartCardUnit:SingleWire:Connect • Printer.SmartCardUnit:SingleWire:Disconnect • Printer.SmartCardUnit:SingleWire:Transmit • Printer.SmartCardUnit:SingleWire:Status • Printer.SmartCardUnit:SingleWire:Control • Printer.SmartCardUnit:SingleWire:GetAttrib	• MagstripeEncode2 • MagstripeRead2 • DoBarcodePark • SmartCardPark • SCardConnect • SCardDisconnect • SCardGetAttrib • SCardStatus • SCardTransmit
Laser engraving	
• Printer.Laser:Engrave:SetupFileName:Set • Printer.Laser:Engrave:Text:Set • Printer.Laser:Engrave:Binary:Set • Printer.Laser:SetupFileName:Get • Printer.Laser:ElementList:Get • Printer.Laser:Upload:File:Get • Printer.Laser:Download:File:Set	Java uses multiple instances of the BidiXPSDriverInterface(*x*) function, where *x* can be the following: • LASER_QUERY_SETUP_FILESLIST • LASER_QUERY_ELEMENT_LIST • LASER_UPLOAD_ZIP_FILE_FROM_PRINTER • LASER_DOWNLOAD_ZIP_FILE_TO_ PRINTER • LASER_ENGRAVE_SETUP_FILE_NAME • LASER_ENGRAVE_TEXT • LASER_ENGRAVE_BINARY
Printer and supplies capabilities and status	
• Printer.PrinterOptions2:Read • Printer.CounterStatus2:Read • Printer.SuppliesStatus3:Read • Printer.ResetCardCount:Set • Printer.Hopper:Status:Get	• GetPrinterOptions2 • GetPrinterCounterStatus2 • GetPrinterSuppliesStatus • ResetCardCounts • GetHopperStatus

IBidiSpl Requests	Java Helper DLL Interface Functions
Laminator	
• Printer.Laminator:BarcodeRead:Set • Printer.Laminator:BarcodeRead:Get • Printer.Laminator:BarcodeReadAndVerify:Set	Not available in Java
Activation	
• Printer.ActivatePrinter:Set	• ActivateOrDisablePrinter
Printer control	
• Printer.ChangePrinterState:Set • Printer.Restart:Set • Printer.PowerDown:Set	• ChangePrinterState • RestartPrinter
Color adjust	
• Printer.AdjustColor:Set • Printer.SetDefaultColor:Set	• SetColorAdjust • DefaultColorAdjust
Lock control	
• Printer.Locks:ChangeLockState:Set • Printer.Locks:ChangePassword:Set	• SetPrinterLockState • ChangeLockPassword

Deprecated IBidiSpl Requests

The following IBidiSpl requests have been deprecated:

- Printer.PrinterOptions:Read was replaced by the following in an earlier version of the driver:
  - Printer.PrinterOptions2:Read
  - Printer.CounterStatus2:Read
  - Printer.SuppliesStatus:Read
- Printer.SuppliesStatus:Read and Printer.SuppliesStatus2:Read were replaced by the following in an earlier version of the driver:
  - Printer.SuppliesStatus3:Read
- The CheckPrintRibbonSupplies and CheckEmbossSupplies options in StartJob. Use the following IBidiSpl request to check the status of remaining supplies using the SDK:
  - Printer.SuppliesStatus3:Read

Order and Timing of Interactive Job Operations

The application must implement the following interactive operations in a specific order or at a specific time:

- A Start Job request is *always* the first operation
- An End Job or Cancel Job request is *always* the last operation
- An End Job request must not be issued until printing operations for the job have entered the driver spooler.

You should perform smart card and interactive magnetic stripe encode and read operations before print operations. Refer to the sample code for examples of best practices regarding the sequence of card operations.

Determine the Success of an IBidiSpl Request

Because all IBidiSpl requests return success, the return value cannot be used to determine the outcome of the request. IBidiSpl requests also return a printer status XML structure. This structure contains information about whether the request succeeded or failed and, if it failed, information about the error that was detected.

For operations that return data from the printer, this structure also contains the data if the operation succeeded.

The following example shows the printer status XML structure returned from a failed StartJob command. The command failed because the printer failed to pick a card.

```
<?xml version="1.0" ?>
<!-- Printer status xml file.-->
<PrinterStatus>
    <ClientID>STATUSTEST</ClientID>
    <WindowsJobID>0</WindowsJobID>
    <PrinterJobID>780</PrinterJobID>
    <ErrorCode>111</ErrorCode>
    <ErrorSeverity>4</ErrorSeverity>
    <ErrorString>Message 111: Card not picked.</ErrorString>
    <DataFromPrinter><![CDATA[ ]]></DataFromPrinter>
</PrinterStatus>
```

The printer status structure contains the following elements:

Element	Description of the element value
ClientID	A unique identifier of the client that created the job. *This element is not used at this time.*
WindowsJobID	The Windows Job ID assigned by the operating system.
PrinterJobID	The Print job ID assigned by the driver.
ErrorCode	If the command succeeded, the ErrorCode is 0 (zero). A non-zero value means an error was detected. For non-zero ErrorCode values, the ErrorSeverity and ErrorString elements also contain values.
ErrorSeverity	Errors are classified into severity levels (1, 2, 3, 4, or 5). The severity level determines which recovery actions are possible.

Element	Description of the element value
ErrorString	A short human-readable description of the error, including the error number. This matches the message that displays on the printer LCD panel.
DataFromPrinter	If the command was intended to read data from the card in the printer and the read operation was a success, this element contains the data in the CDATA section.

Start and End an Interactive Job

To start a job that contains one or more interactive operations, your Visual C++, Visual C#, or VB.NET application must call the IBidiSpl interface with the schema set to `Printer.Print:StartJob:Set`. You can specify input hopper 1 through 6 from which to pick the card. If no hopper is specified, the driver picks a card from hopper 1. You also can specify the side of the card (front or back) that faces up when it is placed in the output hopper

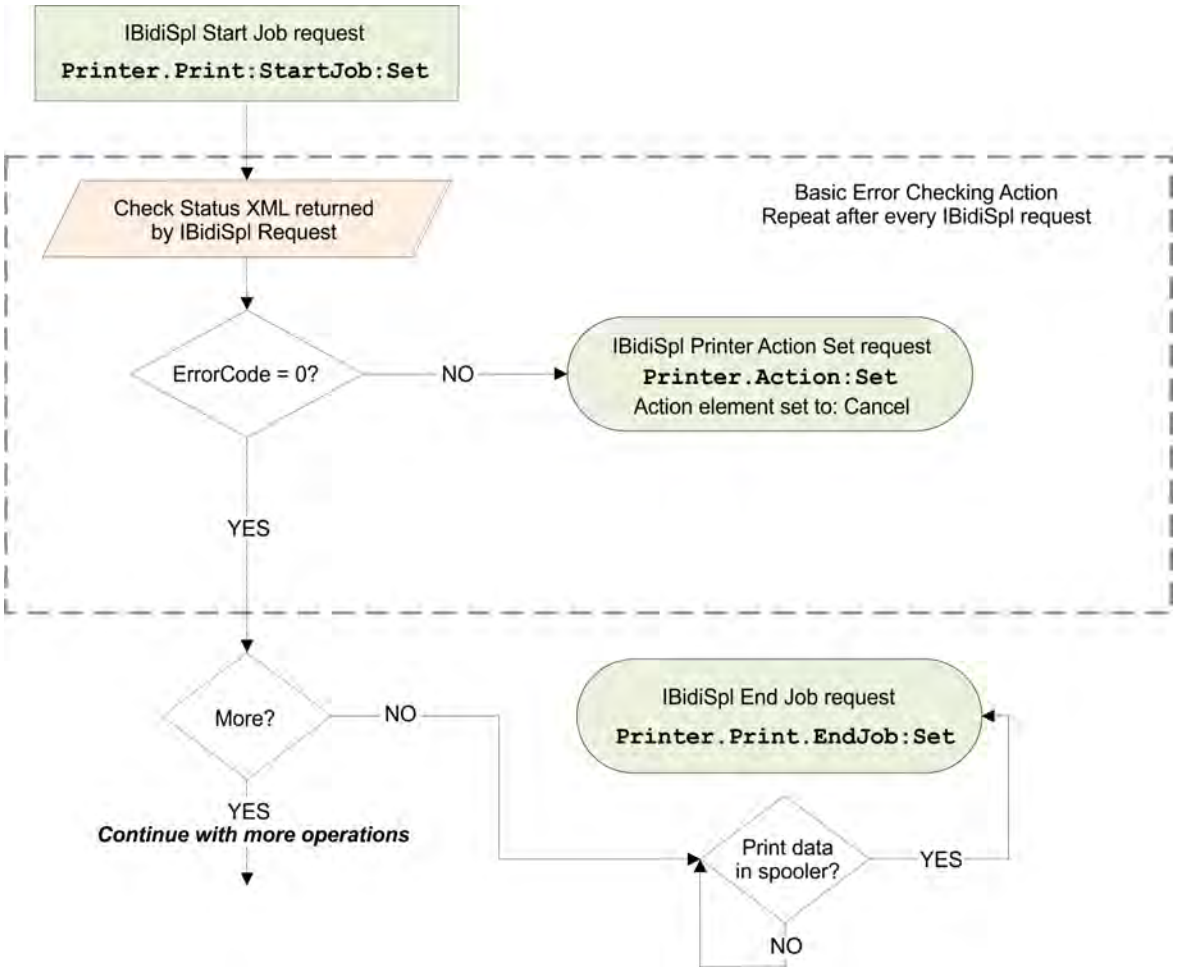
The StartJob request might fail and return error 506. This indicates that the driver or printer is busy and cannot accept another job at this time. A laminating system can have multiple active jobs, and your application might need to wait and retry the StartJob request when the printer is ready to accept it. Refer to the source code samples to see how the StartJob request handles error 506.

For Java, call the StartJob2 method of the dxp01sdk_IBidiSpl_interop.dll.

The start job request always must be the first IBidiSpl request.

To end a job, the Visual C++, Visual C#, or VB.NET application calls the IBidiSpl interface with the schema set to `Printer.Print:EndJob:Set`. For Java, call the EndJob method of the dxp01sdk_IBidiSpl_interop.dll. The end job command is issued after the last interactive operation is successful.

If printing follows the interactive operations, the end job request cannot be sent until the print data appears in the spooler. Submitting an end job immediately results in the job ending before the print data is detected. This results in a second card that contains only the print data. The SDK sample code demonstrates a reliable method for detecting that the print data is in the spooler.



Sample Code

For working code that demonstrates interactive mode Start Job, End Job, and basic error recovery, refer to the following samples:

Visual C++, Visual C#, and VB.NET	magstripe smartcard
Java	magstripe smartcard

Get the Status of an Interactive Job

Your application can retrieve the status for the current interactive job to determine if the printer is still actively processing the card or if the card is complete. The PrinterJobID is used to identify the job. This ID is part of the Printer Status structure returned from the Start Job request.

To retrieve job status, your application uses the IBidiSpl interface with the schema set to `Printer.JobStatus:Read` to send an XML structure with the Printer Job ID of the current interactive job. For Java, call the GetJobStatusXML method of the Java helper DLL (dxp01sdk_IBidiSpl_interop.dll).

```
<?xml version=\"1.0\"?>
<!--job status xml-->
<JobStatus>
    <PrinterJobID>5860</PrinterJobID>
</JobStatus>
```

The Job Status request returns the job status in another XML structure.

```
<?xml version="1.0" ?>
<!-- Job status xml file. -->
<JobStatus>
    <ClientID>STATUSTEST</ClientID>
    <WindowsJobID>5</WindowsJobID>
    <PrinterJobID>5680</PrinterJobID>
    <JobState>JobActive</JobState>
    <JobRestartCount>0</JobRestartCount>
</JobStatus>
```

The ClientID, WindowsJobID, and PrinterJobID have the same meaning as the Printer Status elements returned from other IBidiSpl requests. The JobState and JobRestartCount are unique to this request.

Element	Description of the element value
JobState	The state of the job. The value is one of the following: JobActive, JobSucceeded, JobFailed, JobCancelled, or NotAvailable. Kiosk printers have two additional states: CardReadyToRetrieve and CardNotRetrieved.
JobRestartCount	The number of times the job was retried. This is always zero for interactive jobs.

Using the JobState value, your application can determine if the card is still being processed by the printer or, if it has completed, whether it was personalized successfully.

If your application sends jobs to a kiosk printer, the JobState value indicates if the card is ready to be retrieved, or was not retrieved and moved to the reject tray. Your application needs to poll for job completion after submitting the job until the JobState value is returned. (Refer to Appendix E: "SDK CE870 Kiosk System Support" for information about how the SDK works with a kiosk system.)

JobState value	Description
JobActive	A card is still being personalized by the printer.
JobSucceeded	The card is complete. The job completed without a detected error.
JobFailed	The card is complete. An error forced the job to terminate before the card personalization process completed.
JobCancelled	The card is complete. The job was canceled before the card personalization process completed.
NotAvailable	There is no information for the PrinterJobID provided. Either the value provided is wrong or this is no longer the current job.
CardReadyToRetrieve	The card is complete. The user can retrieve the card from the printer. This support is available for kiosk printers only.
CardNotRetrieved	The card is complete. The printer moved the card to the reject tray because it was not retrieved in time by the user. This support is available for kiosk printers only.

Sample Code

For working code that demonstrates interactive mode Job Status use, refer to the following samples:

Visual C++, Visual C#, and VB.NET	magstripe smartcard status
Java	magstripe smartcard printer_status

Interactive Mode Error Recovery

When the driver is in interactive mode, errors are reported back to your application through the printer status structure returned by every IBidiSpl request. Your application also can get this information by calling the IBidiSpl interface with the schema set to `Printer.PrintMessages:Read`. For Java, call the PrinterStatusXML.GetPrinterMessages() method of the Java helper DLL (dxp01sdk_IBidiSpl_interop.dll).

Error-Related Values in the Printer Status Structure

Three values in the Printer Status structure are used to communicate error information to your application.

Element	Description of the element value
ErrorCode	If the command succeeded, the ErrorCode is 0 (zero). A non-zero value means an error was detected. The value of the ErrorCode element will be one of the message numbers listed in Appendix A: "Error Description Strings". For non-zero ErrorCode values, the ErrorSeverity and ErrorString elements also contain values.
ErrorSeverity	Errors are classified into severity levels (1, 2, 3, 4, or 5). The severity level determines which error recovery actions are possible.
ErrorString	Contains a short description of the error, including the error number. Appendix A: "Error Description Strings" lists the ErrorString values your application can receive from the driver while in interactive mode. The ErrorString value is in the language of the operating system if the language is one of the translations released with the driver.

ErrorSeverity	Severity description	Action
1	**Alert**—Unrecoverable issue for job	Cancel job
2	**Critical**—Unrecoverable issue for job	Cancel job
3	**Error**—Unrecoverable issue for card; recoverable issue for job	Restart or cancel job
4	**Warning**—Recoverable issue for card	Resume or cancel job
5	**Notice**—Information only	None required

Recovery from Errors

To clear an error while in interactive mode, your application uses the IBidiSpl interface with the schema set to `Printer.Action:Set` to send an XML structure with the Printer Job ID of the current interactive job, the ErrorCode you are responding to, and the action you want to take. Java can call the CancelJob or ResumeJob method of the Java helper DLL (dxp01sdk_IBidiSpl_interop.dll).

You must set the ErrorCode to match the error you are responding to for successful error recovery.

The following example shows the structure sent to cancel a job when the input hopper is empty.

```
<?xml version="1.0"?>
<!--printer command xml-->
<PrinterAction>
    <Action>100</Action>
    <PrinterJobID>5860</PrinterJobID>
    <ErrorCode>112</ErrorCode>
</PrinterAction>
```

Action value	Action description	Allowed for ErrorSeverity level
100	**Cancel**—Reject the current card. End the current job.	All
101	**Resume**—Attempt to continue with the current card.	4

Basic Error Recovery (Recommended)

The most robust form of error recovery from an interactive mode error is to cancel the job. Using this error recovery strategy, your application reports the job as failed and, if a card has been picked, it is ejected from the printer. After you correct the cause of the error, you can attempt the card personalization job again.

Advanced Error Recovery

By evaluating the ErrorSeverity value, your application sometimes can offer to resume the job after the cause of the error is corrected. In practice, this complicates error recovery because the application must poll the driver for printer status in the event that the error is corrected and cleared using the printer LCD display. If the ErrorCode is 0, the application can assume that the error was cleared using the printer LCD.

Cancel All Jobs

If you know that your application is the only one sending jobs to the printer, you can cancel all the jobs in the printer to return it to a known good state. This is not recommended for production use, but can be helpful during development.

A laminating system can have multiple active jobs. Using Cancel All Jobs also cancels jobs that are not in an error state.

Sample Code

For working code that demonstrates how to cancel all jobs, refer to the following samples:

Visual C++, Visual C#, and VB.NET	printer_control
Java	printer_control

Errors Cleared at the Printer

After an error condition is corrected at the printer, the operator sometimes can use either the application or the printer's front panel to report that the error is corrected. We recommend that operators be instructed to use the application to acknowledge that error conditions are corrected. Otherwise, the application may get out of sync with the state of the printer.

Suppress the Driver Message Display

If you prefer to have your application manage error reporting and resolution, you can configure the driver to suppress the display of messages. Refer to Appendix D: "Suppress the Driver Message Display" for details.

Encode a Magnetic Stripe with Data

There are three ways to encode data onto a magnetic stripe on the a card.

- Use magnetic stripe escapes in the card data to instruct the driver to encode an IAT track; the data is included between the escape characters. This is processed by the driver along with the print data and does not require interactive mode. Refer to the "Magnetic Stripe Escapes" section of the *XPS Card Printer Driver User's Guide* for details about how to use escapes for magnetic stripe encoding.
- Use the magnetic stripe fonts installed with the Card Printer Driver to encode IAT or JIS formatted data by placing the data on the card design and specifying the magnetic stripe font for the format and track desired. This is processed by the driver along with the print data and does not require interactive mode. Refer to the "Magnetic Stripe Fonts" section of the *XPS Card Printer Driver User's Guide* for details about how to use magnetic stripe fonts for magnetic stripe encoding.
- Use the IBidiSpl interface to pass magnetic stripe data through the driver in the format expected by the printer. This method is described in the following sections.

The printer must be configured to match the format of the magnetic stripe data being sent.

Interactive Mode Magnetic Stripe Encoding

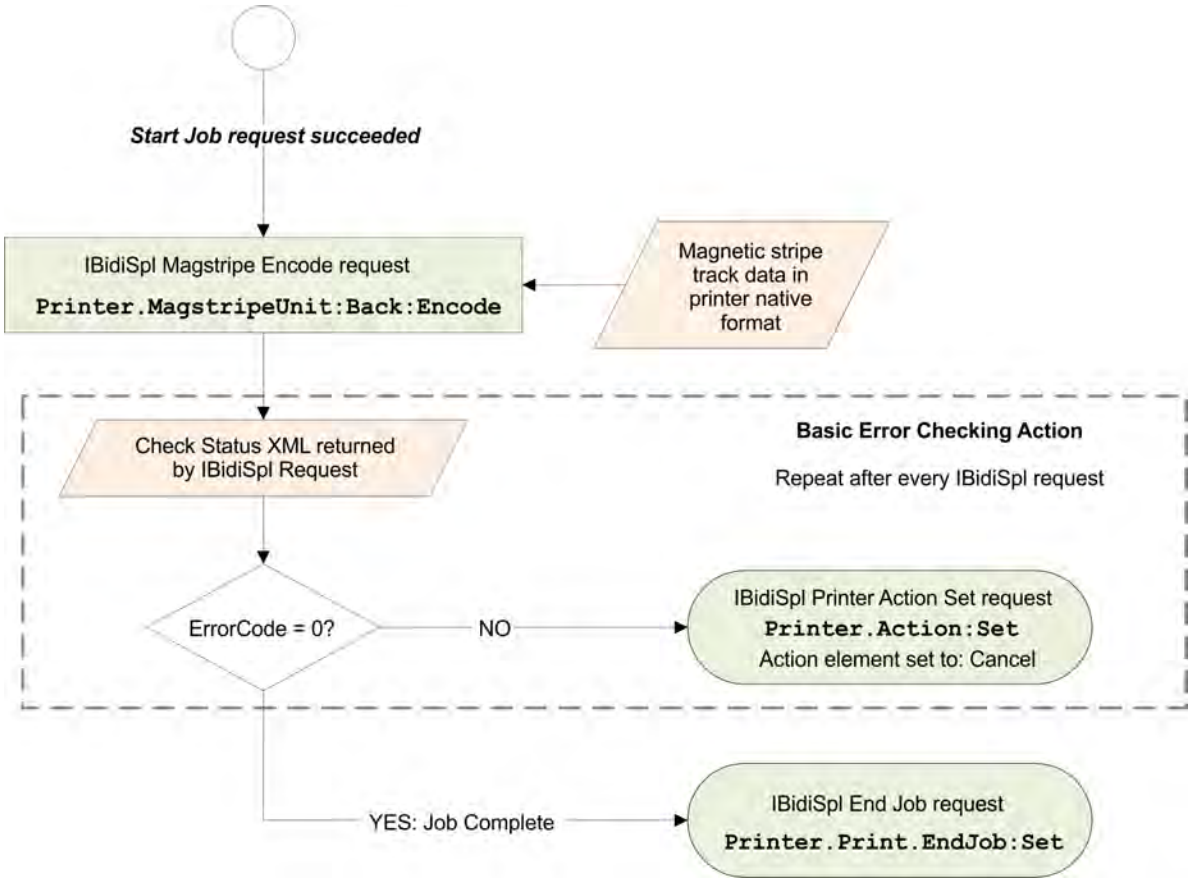
Using the IBidiSpl interface, a card's magnetic stripe can be encoded on the front side or back side of the card. The following assumes you are encoding to the back side of the card.

To encode a magnetic stripe with data, your application calls the IBidiSpl interface with the schema set to `Printer.MagstripeUnit:Back:Encode`. For Java, call the MagstripeEncode2 method of the Java helper DLL (dxp01sdk_IBidiSpl_interop.dll).

The IBidiSpl commands used to encode only the magnetic stripe on a card are:

1. **StartJob**—The printer starts the job and picks the card.
2. **MagstripeEncode**—The application sends the magnetic stripe track data.
3. **EndJob**—The printer ejects the card into the output hopper.

The following flowchart illustrates magnetic stripe encoding:



Magnetic Stripe Track Data Format

When using interactive mode magnetic stripe encoding, the magnetic stripe track data must be provided in the XML format the printer expects. The track data itself must be encoded as UTF-8 and then converted to base64 ASCII. Your application also is responsible for sending track data that is valid for the magnetic stripe format configured at the printer.

The following example shows an XML structure with three tracks of IAT data: track 1 = TRACK1, track 2 = 1122, track 3 = 321.

```
<?xml version="1.0" encoding="UTF-8"?>
<magstripe >
    <track number="1">
        <base64Data>VFJBQ0sx</base64Data>
    </track>
    <track number="2">
        <base64Data>MTEyMg==</base64Data>
    </track>
    <track number="3">
        <base64Data>MzIx</base64Data>
    </track>
</magstripe >
```

Sample Code—Magnetic Stripe Encode

For working code that demonstrates interactive mode magnetic stripe encoding, refer to the following samples:

Visual C++, Visual C#, and VB.NET	magstripe
Java	magstripe

Read Data From a Magnetic Stripe

Using the IBidiSpl interface, data can be read from the tracks of a card's magnetic stripe on the back side of the card. To read data from the magnetic stripe, your application calls the IBidiSpl interface with the schema set to `Printer.MagstripeUnit:Back:Read`. For Java, call the MagstripeRead2 method of the Java helper DLL (dxp01sdk_IBidiSpl_interop.dll).

Like all IBidiSpl requests, the printer status XML structure is returned to your application. The magnetic stripe track data is returned inside the CDATA element of the printer status structure. This data comes directly from the printer without any modification from the driver.

```
<?xml version="1.0"?>
<!--Printer status xml file.-->
<PrinterStatus>
    <ClientID>STATUSTEST_{200AEAAC-CA0A-4AF6-BD77-083A5836AE1A}</ClientID>
    <WindowsJobID>0</WindowsJobID>
    <PrinterJobID>5837</PrinterJobID>
    <ErrorCode>0</ErrorCode>
    <ErrorSeverity>0</ErrorSeverity>
    <ErrorString></ErrorString>
    <DataFromPrinter><![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<magstripe xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope" xmlns:SOAP-
ENC="http://www.w3.org/2003/05/soap-encoding" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:DPCLMagStripe="urn:dpcl:magstripe:2010-01-19" xsi:type="DPCLMagStripe:MagStripe"
SOAP-ENV:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
    <track number="1">
        <base64Data>zw9PkBBQQzw9PkBBQUVJTVFVWV1hZWltcXV5fICEiIyQlJicoKSorLA==</
        base64Data>
    </track>
    <track number="2">
        <base64Data>MDEyMzQ1Njc4OTo7PD0+jc4OTo7PD0+MDEyMzQ1Ng==</base64Data>
    </track>
    <track number="3">
        <base64Data>MDEyMzQ1Njc4OTo7PDDEyMzQ1Njc4OTo7PD0+MDEyMzQ1Njc4OTo7PD0=</
        base64Data>
    </track>
</magstripe>]]></DataFromPrinter>
</PrinterStatus>
```

The track data must be converted from base64 ASCII to the format required by your application.

For example, a job consisting of magnetic stripe read, magnetic stripe encode, and printing would use the following operations in the order specified:

1. **Start Job**—The printer starts the job and picks the card.
2. **Magnetic Stripe Read**—The application reads the magnetic stripe track data.
3. **Magnetic Stripe Encode**—The application sends the magnetic stripe track data.
4. **Print card side(s)**—Use the Windows printing interface (for example, GDI, WinForms), not IBidiSpl.

5. Wait for the print data to enter the spooler.
6. **End Job**—The printer completes printing and then ejects the card into the output tray.

Sample Code—Magnetic Stripe Read

For working code that demonstrates interactive mode magnetic stripe read, refer to the following samples:

Visual C++, Visual C#, and VB.NET	magstripe
Java	magstripe

Place a Card in the Bar Code Reader

If the printer is equipped with the optional bar code reader, the IBidiSpl interface allows you to park a pre-serialized card in the reader, wait for the bar code on the card to be read, and then move the card out of the reader. To park the card in the bar code reader, your application calls the IBidiSpl interface with the schema set to either `Printer.BarcodeUnit:Front:Park` or `Printer.BarcodeUnit:Back.Park`. For Java, call the DoBarcodePark method of the Java helper DLL (dxp01sdk_IBidiSpl_interop.dll).

After the bar code read completes, your application controls if the card is placed in the reject tray, or if it continues on to other personalization operations. To resume or cancel the job, use the IBidiSpl interface with the schema set to `Printer.Action:Set` to send an XML structure with the Printer Job ID of the current interactive job and the action you want to take. Java can call the CancelJob or ResumeJob method of the Java helper DLL.

```
<?xml version="1.0"?>
<!--printer command xml-->
<PrinterAction>
    <Action>101</Action>
    <PrinterJobID>6524</PrinterJobID>
    <ErrorCode>0</ErrorCode>
</PrinterAction>
```

A Resume action (Action value = 101) indicates that the bar code read completed successfully, and the card is ready for further processing.

A Cancel action (Action value = 100) indicates that the bar code read failed, and the card should be rejected without any further personalization.

For example, a job consisting of bar code read and printing would use the following operations in the order specified:

1. **StartJob**—The printer starts the job and picks the card.
2. **ParkCard**—The printer parks the card at the bar code station.

3. **ResumeJob**—The printer moves the card from the bar code station so that the card can be processed further.
4. **Print Card Side(s)**—Use the Windows printing interface (for example, GDI, WinForms), not IBidiSpl.
5. Wait for the print data to enter the spooler.
6. **EndJob**—The printer completes printing and then ejects the card into the output tray.

Use the SDK command BARCODE_PARK when the bar code is on the front side of the card or BARCODE_PARK_BACK when a bar code is on the back side of the card.

For printers running D3 firmware, set the Printer Manager BarcodeLocation setting to CardFront and use the SDK BARCODE_PARK and BARCODE_PARK_BACK commands to specify the card side to read.

Read Data from a Serialized Laminate Bar Code

If your system includes a CLM laminator that is equipped with the optional bar code scanner, the IBidiSpl interface allows you to read data from the bar code printed on the serialized overlay material. To read the bar code data, your application calls the IBidiSpl interface with the schema set to `Printer.Laminator:BarcodeRead:Get`.

The application also can let the printer know whether or not the bar code data will be verified by calling the IBidiSpl interface with the schema set to one of the following: `Printer.Laminator:BarcodeRead:Set` or `Printer.Laminator:BarcodeReadAndVerify:Set.`

The `Printer.Laminator:BarcodeRead:Set` command simply retrieves the bar code data and the card continues automatically. When you use the `Printer.Laminator:BarcodeRead AndVerify:Set` command, the printer stops after the bar code data is returned and waits for the application to instruct it to continue or to reject the card.

The bar code read commands differ somewhat from other commands in that the act of reading the bar code in the laminator occurs after the card is printed. Thus, your application makes the request to read the bar code and then must wait and check for the data to be returned. The driver SDK interface allows you to specify a value for the wait time, or to allow an infinite wait time (this is the default). We recommend that your application does not specify a timeout value. This gives the laminator time to warm up, which can take up to several minutes if it is just starting, before it accepts the card for processing.

You also have the option to save the bar code read results to a file.

Sample Code—Serialized Laminate Bar Code Read

For working code illustrating best practices for the serialized laminate bar code read, refer to the following samples:

Visual C++, Visual C#, and VB.NET	lamination_barcode_read
Java	Java does not support this feature.

Place a Card in the Smart Card Station

Using the IBidiSpl interface, a card can be placed (parked) in the printer's smart card station where it can be read, personalized, or both. To park a card in the printer's smart card station, your application calls the IBidiSpl interface with the schema set to `Printer.SmartCardUnit:Front:Park` or `Printer.SmartCardUnit:Back:Park`. For Java, call the SmartCardPark method of the Java helper DLL (dxp01sdk_IBidiSpl_interop.dll).

After smart card personalization completes, your application controls if the card is placed in the reject tray, or if it continues on to other personalization operations. To resume or cancel the job, use the IBidiSpl interface with the schema set to `Printer.Action:Set` to send an XML structure with the Printer Job ID of the current interactive job and the action you want to take.

```
<?xml version="1.0"?>
<!--printer command xml-->
<PrinterAction>
    <Action>101</Action>
    <PrinterJobID>5860</PrinterJobID>
    <ErrorCode>0</ErrorCode>
</PrinterAction>
```

A Resume action (Action value = 101) indicates that smart card personalization completed successfully, and the card is ready for further processing.

A Cancel action (Action value = 100) indicates that smart card personalization failed, and the card should be rejected without any further personalization.

For Java, call either the ResumeJob, CancelJob, or EndJob method of the Java helper DLL (dxp01sdk_IBidiSpl_interop.dll).

For example, a job consisting of smart card encoding and printing would use the following operations in the order specified:

1. **StartJob**—The printer starts the job and picks the card.
2. **ParkCard**—The printer parks the card at the smart card station.
3. **ResumeJob**—The printer moves the card from the smart card station so that the card can be processed further.

4. **Print Card Side(s)**—Use the Windows printing interface (GDI, WinForms, etc.), not IBidiSpl.
5. Wait for the print data to enter the spooler.
6. **EndJob**—The printer completes printing and then ejects the card into the output tray.

Sample Code—Smart Card Park

For working code that demonstrates interactive mode smart card station park, refer to the following samples:

Visual C++, Visual C#, and VB.NET	smartcard
Java	smartcard

Personalize a Smart Card

If your printer is equipped with a single-wire smart card reader, you can personalize the card using the driver SDK after the smart card is parked. The IBidiSpl requests used to do this are:

- Printer.SmartCardUnit:SingleWire:Connect
- Printer.SmartCardUnit:SingleWire:Disconnect
- Printer.SmartCardUnit:SingleWire:Transmit
- Printer.SmartCardUnit:SingleWire:Status
- Printer.SmartCardUnit:SingleWire:GetAttrib

Printer.SmartCardUnit:SingleWire:Connect

A Connect request establishes a connection between the calling application and a smart card parked in the reader. If no card exists in the reader, an error is returned.

To connect to the smart card in the reader, use the IBidiSpl interface with the schema set to `Printer.SmartCardUnit:SingleWire:Connect`. For Java, call the SCardConnect method of the Java helper DLL (dxp01sdk_IBidiSpl_interop.dll).

Smart Card Connect Request—Required Information

Your application must create an XML structure indicating the protocol to use (contact or contactless). The driver receives this XML formatted data as a BIDI_BLOB.

```
<?xml version=\"1.0\"?>
<SmartcardConnect>
    <PreferredProtocol>SCARD_PROTOCOL_CL</PreferredProtocol>
</SmartcardConnect>
```

Protocol Name Value	Connection Type
SCARD_PROTOCOL_CL	Contactless
SCARD_PROTOCOL_T0_OR_T1	Contacted

Smart Card Connect Request—Return Values

- The IBidiSpl interface returns a printer status XML structure. The printer status includes a valid ClientID, WindowsJobID (if applicable, 0 for interactive mode jobs), PrinterJobID, and ErrorCode.
  - If the ErrorCode is zero, the connection request was successful.
  - If the ErrorCode is non-zero, the connection request failed. In this case, the printer status XML file also contains values for ErrorSeverity and ErrorString.
- The CDATA section in the printer status XML structure returns any response from the smart card reader.

Smart Card Connect Request—Status Returned

The following example shows a printer status XML structure returned by a single-wire smart card Connect IBidiSpl request. The smart card reader response is included in the CDATA section.

```
<?xml version="1.0"?>
<!--Printer status xml file.-->
<PrinterStatus>
<ClientID>Test-Win7_{716DD9A0-CF52-4176-B1C0-A10FA8DB055A}</ClientID>
<WindowsJobID>0</WindowsJobID>
<PrinterJobID>6049</PrinterJobID>
<ErrorCode>0</ErrorCode>
<ErrorSeverity>0</ErrorSeverity>
<ErrorString></ErrorString>
<DataFromPrinter><![CDATA[
<?xml version="1.0"?><!--smartcard response xml-->
<SmartcardResponse>
<Protocol>SCARD_PROTOCOL_RAW</Protocol>
<State> </State>
<Status>SCARD_S_SUCCESS</Status>
<Base64Data> </Base64Data>
</SmartcardResponse>
  ]]></DataFromPrinter></PrinterStatus>
```

Printer.SmartCardUnit:SingleWire:Disconnect

A Disconnect request terminates a connection previously opened between the calling application and a smart card in the reader.

To terminate a connection, use the IBidiSpl interface with the schema set to `Printer.SmartCardUnit:SingleWire:Disconnect`. For Java, call the SCard Disconnect method of the Java helper DLL (dxp01sdk_IBidiSpl_interop.dll).

Smart Card Disconnect Request—Required Information

Your application must create an XML structure indicating the disconnect method to use. The driver receives this XML formatted data as a BIDI_BLOB.

```
<?xml version=\"1.0\"?>
<SmartcardDisconnect>
    <Disposition>SCARD_LEAVE_CARD</Disposition>
</SmartcardDisconnect
```

Disconnect Method Value	Action
SCARD_LEAVE_CARD	Leave as is
SCARD_RESET_CARD	Reset the card
SCARD_UNPOWER_CARD	Power down the card

Smart Card Disconnect Request—Return Values

- The IBidiSpl interface returns a printer status XML structure. The printer status includes a valid ClientID, WindowsJobID (if applicable, 0 for interactive mode jobs), PrinterJobID and ErrorCode.
  - If the ErrorCode is zero the request was successful.
  - If the ErrorCode is non-zero the request failed. In this case, the printer status XML file also contains values for ErrorSeverity and ErrorString.
- The CDATA section in the printer status XML structure returns any response from the smart card reader.

Smart Card Disconnect Request—Status Returned

The following example shows a printer status XML structure returned by a single-wire smart card Disconnect IBidiSpl request. The single-wire smart card reader response is included in the CDATA section.

```
Sample XML file returned for disconnect
<?xml version="1.0"?>
<!--Printer status xml file.-->
<PrinterStatus>
<ClientID>Test-Win7_{716DD9A0-CF52-4176-B1C0-A10FA8DB055A}</ClientID>
<WindowsJobID>0</WindowsJobID>
<PrinterJobID>6049</PrinterJobID>
<ErrorCode>0</ErrorCode>
<ErrorSeverity>0</ErrorSeverity>
<ErrorString></ErrorString>
<DataFromPrinter><![CDATA[
<?xml version="1.0"?><!--smartcard response xml-->
<SmartcardResponse>
<Protocol> </Protocol>
<State> </State>
<Status>SCARD_S_SUCCESS</Status>
<Base64Data> </Base64Data>
</SmartcardResponse>
]]></DataFromPrinter></PrinterStatus>
```

Printer.SmartCardUnit:SingleWire:Transmit

A Transmit request sends a service request to the smart card and expects to receive data back from the card.

To send a request, use the IBidiSpl interface with the schema set to `Printer.SmartCardUnit:SingleWire:Transmit`. For Java, call the SCardTransmit method of the Java helper DLL (dxp01sdk_IBidiSpl_interop.dll).

Smart Card Transmit Request—Required Information

Your application must create a smart card transmit XML structure with the chip data encoded as Base64 ASCII. The driver receives this XML formatted data as a BIDI_BLOB.

```
<?xml version=\"1.0\"?>
<SmartcardTransmit>
    <SendBuffer>AKQAAA==</SendBuffer>
</SmartcardTransmit>
```

Smart Card Transmit Request—Return Values

- The IBidiSpl interface returns a printer status XML structure. The printer status includes a valid ClientID, WindowsJobID (if applicable, 0 for interactive mode jobs), PrinterJobID, and ErrorCode.
    - If the ErrorCode is zero, the transmit request was successful.
    - If the ErrorCode is non-zero, the transmit request failed. In this case, the printer status XML file also contains values for ErrorSeverity and ErrorString.
- The CDATA section in the printer status XML structure returns any response from the smart card reader.

Smart Card Transmit Request—Status Returned

The following example shows a printer status XML structure returned by a single-wire smart card Transmit IBidiSpl request. The single-wire smart card reader response is included in the CDATA section.

```
<?xml version="1.0"?>
<!--Printer status xml file.-->
<PrinterStatus>
<ClientID>agarwas-Win7_{716DD9A0-CF52-4176-B1C0-A10FA8DB055A}</ClientID>
<WindowsJobID>0</WindowsJobID>
<PrinterJobID>6049</PrinterJobID>
<ErrorCode>0</ErrorCode>
<ErrorSeverity>0</ErrorSeverity>
<ErrorString></ErrorString>
<DataFromPrinter><![CDATA[
<?xml version="1.0"?><!--smartcard response xml-->
<SmartcardResponse>
<Protocol> </Protocol>
<State> </State>
<Status>SCARD_S_SUCCESS</Status>
<Base64Data>ZwA=</Base64Data>
</SmartcardResponse>
]]></DataFromPrinter></PrinterStatus>
```

Printer.SmartCardUnit:SingleWire:Status

A Status request provides the current status of the smart card in the reader. You can call it any time after a successful call to SCardConnect and before a successful call to SCardDisconnect. It does not affect the state of the reader or reader driver.

To retrieve the smart card status, use the IBidiSpl interface with the schema set to `Printer.SmartCardUnit:SingleWire:Status`. For Java, call the SCardStatus method of the Java helper DLL (dxp01sdk_IBidiSpl_interop.dll).

Smart Card Status Request—Return Values

- The IBidiSpl interface returns a printer status XML structure. The printer status includes a valid ClientID, WindowsJobID (if applicable, 0 for interactive mode jobs), PrinterJobID, and ErrorCode.
  - If the ErrorCode is zero, the status request was successful.
  - If the ErrorCode is non-zero, the status request failed. In this case, the printer status XML file also contains values for ErrorSeverity and ErrorString.
- The CDATA section in the printer status XML structure returns any response from the smart card reader.

Smart Card Status Request—Status Returned

The following example shows a sample printer status XML structure returned by a single-wire smart card Status IBidiSpl request. The single-wire smart card response is included in the CDATA section.

```
<?xml version="1.0"?>
<!--Printer status xml file.-->
<PrinterStatus>
    <ClientID>agarwas-Win7_{716DD9A0-CF52-4176-B1C0-A10FA8DB055A}</ClientID>
    <WindowsJobID>0</WindowsJobID>
    <PrinterJobID>6049</PrinterJobID>
    <ErrorCode>0</ErrorCode>
    <ErrorSeverity>0</ErrorSeverity>
    <ErrorString></ErrorString>
    <DataFromPrinter><![CDATA[
<?xml version="1.0"?><!--smartcard response xml-->
<SmartcardResponse>
<Protocol>SCARD_PROTOCOL_RAW</Protocol>
<State>SCARD_PRESENT|SCARD_POWERED|SCARD_NEGOTIABLE</State>
<Status>SCARD_S_SUCCESS</Status>
<Base64Data>O/2RAP+RgXH+QABCAAAAAACBgYAXCACIGQ==</Base64Data>
</SmartcardResponse>
]]></DataFromPrinter></PrinterStatus>
```

Printer.SmartCardUnit:SingleWire:GetAttrib

A GetAttrib request retrieves the current reader attributes. It does not affect the state of the reader, driver, or card.

To retrieve the smart card reader attributes, use the IBidiSpl interface with the schema set to `Printer.SmartCardUnit:SingleWire:GetAttrib`.

Smart Card GetAttrib Request—Required Information

Your application must create a smart card status XML structure with the name of the reader attribute you want information for. The driver receives this XML formatted data as a BIDI_BLOB.

```
<?xml version=\"1.0\"?>
<!--smartcard get attrib xml-->
<SmartcardGetAttrib>
       <Attr>SCARD_ATTR_VENDOR_IFD_VERSION</Attr>
</SmartcardGetAttrib>
```

AttribName	Action
SCARD_ATTR_VENDOR_NAME	Reader Vendor
SCARD_ATTR_VENDOR_IFD_ VERSION	Vendor-supplied interface device version. DWORD is in the form 0xMMmmbbbb, where: • MM = major version • mm = minor version • bbbb = build number
SCARD_ATTR_VENDOR_IFD_TYPE	Vendor-supplied interface device type (model designation of reader)
SCARD_ATTR_VENDOR_IFD_SERIAL_ NO	Vendor-supplied interface device serial number

Smart Card GetAttrib Request—Return Values

- The IBidiSpl interface returns a printer status XML structure. The printer status includes a valid ClientID, WindowsJobID (if applicable, 0 for interactive mode jobs), PrinterJobID, and ErrorCode.
  - If the ErrorCode is zero, the GetAttrib request was successful.
  - If the ErrorCode is non-zero, the GetAttrib request failed. In this case, the printer status XML file also contains values for ErrorSeverity and ErrorString.
- The CDATA section in the printer status XML structure returns any response from the smart card reader.

Smart Card GetAttrib Request—Status Returned

The following is an example of a printer status XML structure returned by a single-wire smart card GetAttrib IBidiSpl request. The single-wire smart card response is included in the CDATA section. In this case, it is a request for the vendor name. The name is returned in the Base64Data element as Base64 encoded ASCII and must be decoded by your application.

```
<?xml version="1.0"?>
<!--Printer status xml file.-->
<PrinterStatus>
<ClientID>agarwas-Win7_{716DD9A0-CF52-4176-B1C0-A10FA8DB055A}</ClientID>
<WindowsJobID>0</WindowsJobID>
<PrinterJobID>6049</PrinterJobID>
<ErrorCode>0</ErrorCode>
<ErrorSeverity>0</ErrorSeverity>
<ErrorString></ErrorString>
<DataFromPrinter><![CDATA[
<?xml version="1.0"?><!--smartcard response xml-->
<SmartcardResponse>
<Protocol> </Protocol>
<State></State>
<Status>SCARD_S_SUCCESS</Status>
<Base64Data> O/2RAP+RgXH+QABCAAAAAACBgYAXCACIGQ==</Base64Data>
</SmartcardResponse>
]]></DataFromPrinter></PrinterStatus>
```

Sample Code—Single-Wire Smart Card Personalization

For working code that demonstrates personalization of a smart card, refer to the following samples:

Visual C++, Visual C#, and VB.NET	smartcard_singlewire
Java	smartcard_singlewire

The SDK sample code wraps the IBidiSpl interface providing an interface that is similar to the Microsoft Windows SCard API. You can include this code in your application or communicate directly to the IBidiSpl interface, as you prefer.

Return Values from the Sample Code SCard Wrapper

Return values are provided by the printer as strings, but PC/SC applications expect a numeric HRESULT value. The SDK wrapper code converts the return string to the HRESULT value expected by the application. Possible return values are either SCARD_S_SUCCESS or an error. You can find PC/SC error code information at: http://msdn.microsoft.com/en-us/library/ms936965.aspx

Read and Write Data to MIFARE Classic over Single-Wire

The SDK smartcard_singlewire_duali_mifare sample demonstrates how to read and write data to a MIFARE Classic chip using Duali smart card reader commands over a single-wire smart card connection.

For working code that demonstrates personalization of a MIFARE Classic smart card, refer to the following samples:

Visual C++, Visual C#, and VB.NET	smartcard_singlewire_duali_mifare
Java	Java does not support this feature.

Read and Writer Data to an Omnikey Reader over Single-Wire

The SDK smartcard_singlewire_omnikey sample demonstrates how to read and write data to a smart card chip using the Omnikey smart card reader. The Omnikey reader supports multiple cards types, allowing you to select the type of card to use from the following: Mifare, iClass, or HIDProx.

For working code that demonstrates personalization using an Omnikey reader, refer to the following samples:

Visual C++, Visual C#, and VB.NET	smartcard_singlewire_omnikey
Java	Java does not support this feature .

Application Responsibilities with Single-Wire Smart Card

Your application must be able to do the following:

- Verify that the single-wire smart card reader is available in the printer. You can use the IBidiSpl interface to get the printer options to do this.
- Park the smart card before using the single-wire smart card reader, and move the card out of the reader when the personalization is complete.
- Send data the chip can accept. The driver does not check or alter the data.
- Format the data so it can be understood by the printer and reader.

Applications written for PC/SC readers require modification to use the single-wire smart card feature. The PC/SC interface commonly used to interact with USB-connected smart card readers is not directly supported by the driver API.

Laser Engraving

Your application can send data to the CL900 desktop laser system that can be laser engraved on a card. The laser system requires additional setup and card design information to be present on the laser system itself. Refer to the documentation for your laser system for complete information about setting up the required files.

Retrieve Laser Card Setup Files

Using the IBidiSpl interface, your application can retrieve the laser card setup information in the printer. The information is required to process the laser data.

To retrieve the names of all of the laser card setup files present in printer, your application calls the IBidiSpl interface with the schema set to `Printer.Laser:SetupFileName:Get`.

The following example shows the XML structure that is returned with three laser card setup names.

```
<?xml version="1.0" encoding="UTF-8"?>
<QuerySetupsResult>
    <LaserCardSetups>
        <LaserCardSetup name="TestCardSetup"/>
        <LaserCardSetup name="TestCardSetup1"/>
        <LaserCardSetup name="TestCardSetup2"/>
    </LaserCardSetups>
</QuerySetupsResult>
```

Retrieve Laser Elements in a Setup File

The application also can retrieve the variable laser elements in a laser card setup file.

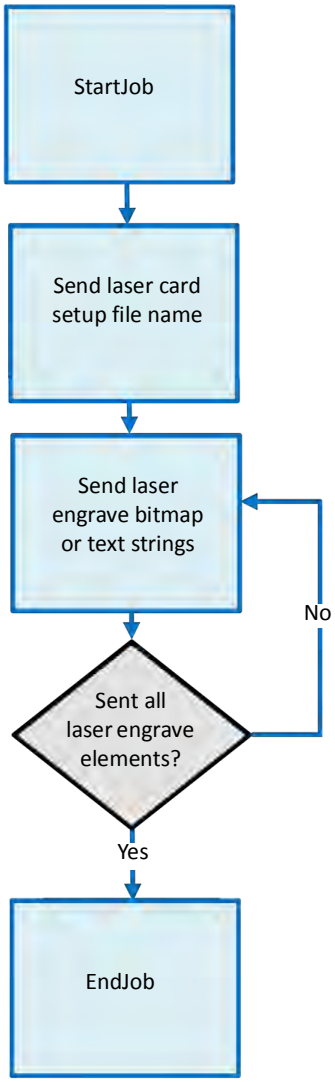
To retrieve element names of a laser card setup file present in the laser system, your application calls the IBidiSpl interface with the schema set to `Printer.Laser:ElementList:Get` with the laser setup file name as data.

The following example shows the XML structure of the TestCardSetup1 laser card setup file. The setup file contains seven variable element names. The information returned for each element includes the element name, the type of element, and the card side.

```
<?xml version="1.0" encoding="UTF-8"?>
<QueryElementsResult>
    <ElementInformationList>
        <ElementInformation name="PHOTO" type="BINARY" side="FRONT" />
        <ElementInformation name="GIVEN_NAME" type="TEXT" side="FRONT" />
        <ElementInformation name="FAMILY_NAME" type="TEXT" side="FRONT" />
        <ElementInformation name="DOB" type="TEXT" side="FRONT" />
        <ElementInformation name="SIGNATURE" type="BINARY" side="FRONT" />
        <ElementInformation name="BARCODE_1D" type="TEXT" side="BACK" />
        <ElementInformation name="BARCODE_2D" type="BINARY" side="BACK" />
    </ElementInformationList>
</QueryElementsResult>
```

Use the Laser Sample

The flowchart illustrates the laser engraving process without magnetic stripe encoding or printing.



The Laser sample program shows how to laser engrave three types of laser layouts:

- **Duplex card.** The card is engraved on both sides with variable data elements.
- **Simplex card.** The card is engraved on one side with variable data elements.
- **Static card.** The card is engraved with predefined data that does not need variable elements. The static card can be either single-sided or double-sided.

The laser engrave data can be one of the following types:

- **TEXT.** A UTF-8 text string. This includes text strings and some bar code formats.
- **BINARY.** Base64-encoded binary data. This includes images and bar codes.

The application must send the name of the laser card setup file (LASER_ENGRAVE_SETUP_FILE_NAME).

If the setup file contains variable elements, the application also must send the variable data for each element to be engraved. (LASER_ENGRAVE_TEXT and LASER_ENGRAVE_BINARY).

The following examples illustrate the operations to laser engrave a card. Refer to the sample program for examples that show how to specify the laser operations.

- A job that laser engraves a static layout would use the following IBidiSpl operations in the order specified:
  a. **Start Job**—The printer starts the job and picks the card.
  b. **Specify the laser setup file**—Specify the static laser setup file name.
  c. **End Job**—The printer ejects the card into the output hopper.

- A job that engraves a duplex card layout with variable data elements would use the following IBidiSpl operations in the order specified:

  a. **Start Job**—The printer starts the job and picks the card.

  b. **Specify the laser setup file**—Specify the duplex laser setup file name, and the variable elements count.

  c. **Specify laser engrave data**—Send text or binary information for all the variable elements in the laser setup file.

  d. **End Job**—The printer ejects the card into the output hopper.

  The laser card layout file specifies the element name location and the card side. The application does not have to specify where the element will be engraved.

- A job that encodes a magnetic strip, prints text (depending on the printer capabilities), and laser engraves data would use the following IBidiSpl operations in the order specified:

  a. **Start Job**—The printer starts the job and picks the card.

  b. **Magnetic Stripe Encode**—The application sends the magnetic stripe track data.

  c. **Print card side(s)**—Use the Windows printing interface (for example, GDI, WinForms), not IBidiSpl.

  d. Wait for the print data to enter the spooler.

  e. **Specify the laser setup file**—Specify the laser setup file name, and the variable elements count.

  f. **Specify laser engrave data**—Send text or binary information for all the variable elements in the laser setup file.

  g. **End Job**—The printer completes printing and then ejects the card into the output tray.

Import or Export Laser Setup Files

Using the IBidiSpl interface, your application can transfer laser card setup files from one laser system to another. To export the laser card setup files from the laser system, the driver returns a zip file containing the setup files to the PC. The application then can import the zip file from the PC to another laser system.

The zip file should not be modified by the application before importing it to another laser system.

To **export** all the files related to a laser card setup from a system, your application calls the IBidiSpl interface with the schema set to `Printer.Laser:Upload:File:Get`.

The driver returns the base64-encoded zip file from the laser system. The zip file contains all the items that are linked to the specified laser setup file name, such as the ConCAD file, laser profiles, pattern match setup, and so on.

To **import** the zip file containing the laser setup to a different laser system, your application calls the IBidiSpl interface with the schema set to `Printer.Laser:Download:File:Set`.

The following XML is returned to your application when exporting or importing zip files from, or to, a system:

```
<?xml version="1.0"?>
<!--laser response xml-->
-<LaserResponse>
    <Status>1</Status>
    <Base64Data> </Base64Data>
</LaserResponse>
```

Status Value	Description
1	Action succeeded
0	Action failed

Installed Printer Status, Supplies Status, and Counter Status

Your application can determine the status of the printer and which options are available in the printer, information about the supplies loaded in the printer, and card counts.

Printer Status Information

To retrieve printer status, your application uses the IBidiSpl interface with the schema set to `Printer.PrinterOptions2:Read`. For Java, call the GetPrinterOptions2 method of the Java helper DLL (dxp01sdk_IBidiSpl_interop.dll).

The expanded list of printer information described in the following section requires Printer.PrinterOptions2:Read.

The following shows a sample printer status XML file returned by this request. The information that is returned depends on the values provided by the printer.

```
<?xml version="1.0"?>
<!--Printer options2 xml file.-->
<PrinterInfo2>
    <PrinterStatus>Ready</PrinterStatus>
    <PrinterAddress>172.16.5.79</PrinterAddress>
    <PrinterModel>CXXXX</PrinterModel>
    <PrinterSerialNumber>C15133</PrinterSerialNumber>
    <PrinterVersion>XX.XX.X-X</PrinterVersion>
    <PrinterMessageNumber>0</PrinterMessageNumber>
    <ConnectionPortType>Network</ConnectionPortType>
    <ConnectionProtocol>Version2Secure</ConnectionProtocol>
    <OptionInputhopper>SingleHopper</OptionInputhopper>
    <OptionMagstripe>ISO</OptionMagstripe>
    <OptionSecondaryMagstripeJIS>None</OptionSecondaryMagstripeJIS>
    <OptionRewritable>None</OptionRewritable>
    <OptionSmartcard>Single wire</OptionSmartcard>
    <OptionDuplex>Auto</OptionDuplex>
    <OptionPrinterBarcodeReader>None</OptionPrinterBarcodeReader>
    <OptionLocks>Installed</OptionLocks>
    <LockState>Locked</LockState>
    <PrintEngineType>DirectToCard_DyeSub</PrintEngineType>
    <PrintHead>Installed</PrintHead>
    <ColorPrintResolution>300x300 | 300x600</ColorPrintResolution>
    <MonochromePrintResolution>300x300 |300x600 |300x1200</MonochromePrintResolution>
    <TopcoatPrintResolution>300x300</TopcoatPrintResolution>
    <ModuleEmbosser>Installed</ModuleEmbosser>
    <EmbosserVersion>E1.1.24-0</EmbosserVersion>
    <Laminator>None</Laminator>
    <TactileImpresser>None</TactileImpresser>
    <Kiosk>None</Kiosk>
    <LaserModule>Installed</LaserModule>
    <LaserVisionRegistration>Installed</LaserVisionRegistration>
    <ObscureBlackPanel>None</ObscureBlackPanel>
</PrinterInfo2>
```

Printer Status

The PrinterStatus element contains the state of the printer at the time of the request. Your application can use this to determine if the printer is online and ready to accept a job.

PrinterStatus Value	Description
Unavailable	The printer is not connected or is powered off.
Ready	The printer is available to accept a job.
Busy	The printer is processing a job.
Paused	The printer has errors or has been paused.
Suspended	The printer's front panel or Print Manager application is being used.
Initialize	The printer is powering up and not ready to accept a job.
Shutdown	The printer is powering down and cannot accept a job.

Printer Information

Element Value	Description
PrinterAddress	The IP address of the printer.
PrinterModel	The model of the printer.
PrinterSerialNumber	The serial number of the printer.
PrinterVersion	The firmware version installed on the printer.

Message Number

The MessageNumber element contains the error number if the printer is in an error state. A value of zero means there is no error. (Refer to Appendix A: "Error Description Strings” for a list of messages.)

Printer Connection Information

Element Value	Description
ConnectionPortType	Identifies the physical connection being used to communicate to the printer. The possible values are: • Network • USB
ConnectionProtocol	Identifies the protocol used to communicate with the printer. The possible values are: • Version1 • Version2 • Version2Secure Version2Secure is required if you want all the data exchanged between the driver and printer to be encrypted.

Printer Options

Element Value	Description
OptionInputhopper	The input hopper configuration for this printer. The values are: • SingleFeed • SingleHopper • SingleHopperWithExceptionSlot • MultiHopper6WithExceptionSlot • HopperAutoDetect **Note:** Printers that support hopper detection return the value HopperAutoDetect for OptionInputHopper. Your application then can use the IBidiSpl operation `Printer.Hopper:Status:Get`, or the Java function GetHopperStatus to get the input hopper status.
OptionMagstripe	The magnetic stripe configuration for this printer. The values are: • None • ISO • JIS

Element Value	Description
OptionRewritable	Identifies if this printer supports rewritable cards. The values are: • None • Installed **Note:** For printers that support rewritable cards, the rewritable feature must be enabled and the printer configured correctly.
OptionSecondaryMagstripeJIS	The secondary magnetic stripe configuration for the printer. The values are: • None • Installed
OptionSmartcard	The smart card configuration for this printer. The values are: • None • Installed • Single wire
OptionDuplex	The duplex configuration for this printer. The values are: • Manual • Auto
OptionPrinterBarcodeReader	Indicates if a bar code reader is installed. The values are: • None • Installed **Note:** The CR805 retransfer printer cannot detect if a bar code reader is installed in the printer. The value None is returned even if the bar code reader is installed.
OptionLocks	The lock configuration for this printer. The values are: • None • Installed
LockState	The lock state if the printer has the lock option installed. The values are: • Locked • Unlocked This element is missing if the OptionLock value is None.
PrintEngineType	The type of printer reporting the information. The values are: • DirectToCard_DyeSub • Retransfer_Pigment

Element Value	Description
PrintHead	Indicates if this printer includes a printhead. (The printer might not have a printhead if you are connected to an emboss-only CE840 system.) The values are: • None • Installed
ColorPrintResolution	The color printing resolutions supported by this printer. This is a list of values separated by a “|” character. The value list may include: • 300x300 • 300x600 • 600x600 (CR805 retransfer card printer only) This element is missing if the PrintHead value is None.
MonochromePrintResolution	The monochrome printing resolutions supported by this printer. This is a list of values separated by a "|" character. The value list may include: • 300x300 • 300x600 • 300x1200 • 600x600 (CR805 retransfer card printer only) This element is missing if the PrintHead value is None.
TopcoatPrintResolution	The topcoat resolution supported by this printer. The values are: • 300x300 • Unknown
ModuleEmbosser	Indicates if this printer includes a CEM embosser. The values are: • None • Installed
EmbosserVersion	The embosser firmware version if the system includes an embosser. The element is missing if the EmbossModule value is None.
Laminator	Indicates if the printer includes a laminator and, if so, whether it has one or two lamination stations. The values are: • None • L1 • L1, L2

Element Value	Description
LaminatorFirmwareVersion	The laminator firmware version if the system includes a laminator. This element is missing if the Laminator value is None.
LaminatorImpresser	Indicates if the laminator includes the card impresser option. The values are: • None • Installed
LaminatorScanner	Indicates if the laminator includes the bar code scanner option. The values are: • None • Installed
TactileImpresser	Indicates if the Tactile Impression Module is installed on the printer. The values are: • None • Installed
Kiosk	Indicates if the system is a kiosk system. The values are: • None • Installed
LaserModule	Indicates if the laser module is installed. The values are: • None • Installed
LaserFirmwareVersion	The laser firmware version if the system is a laser system. This element is missing if the LaserModule value is None.
LaserVisionRegistration	Indicates if the Laser Vision module is installed. The values are: • None • Installed This element is missing if the LaserModule value is None.
ObscureBlackPanel	Indicates if the printer has the ability to obscure the information printed with the K panel. The values are: • None • Installed

Sample Code—Printer Status

For working code that demonstrates printer status, refer to the following samples:

Visual C++, Visual C#, and VB.NET	status
Java	printer_status

Supplies Information

Your application can determine the status of supplies using the IBidiSpl interface with the schema set to `Printer.SuppliesStatus3:Read`. For Java, call the GetPrinterSuppliesStatus method of the Java helper DLL (dxp01sdk_IBidiSpl_interop.dll).

The request returns the supplies status XML file.

```
<?xml version="1.0"?>
<!--Printer Supplies3 xml file.-->
<PrinterSupplies3>
    <PrinterStatus>Ready</PrinterStatus>
    <Printer>
        <PrintRibbon>Installed</PrintRibbon>
        <PrintRibbonType>YMCKT</PrintRibbonType>
        <RibbonRemaining>76</RibbonRemaining>
        <RibbonSerialNumber>E0055000008D355F</RibbonSerialNumber>
        <RibbonLotCode>10-23-16 </RibbonLotCode>
        <RibbonPartNumber>535000003</RibbonPartNumber>
        <RibbonRegionCode>0</RibbonRegionCode>
        <RetransferFilmPartNumber>0</RetransferFilmPartNumber>
        <RetranferFilmPercentRemaining></RetransferFilmPercentRemaining>
        <RetransferFilmSerialNumber></RetransferFilmSerialNumber>
        <RetransferFilmLotCode></RetransferFilmLotCode>
    </Printer>
    <Embosser>
        <IndentRibbon>None</IndentRibbon>
        <IndentRibbonType></IndentRibbonType>
        <IndentRibbonRemaining></IndentRibbonRemaining>
        <TopperRibbon>None</TopperRibbon>
        <TopperRibbonType></TopperRibbonType>
        <TopperRibbonRemaining></TopperRibbonRemaining>
        <TopperRibbonSerialNumber></TopperRibbonSerialNumber>
        <TopperRibbonLotCode></TopperRibbonLotCode>
        <TopperRibbonPartNumber></TopperRibbonPartNumber>
    </Embosser>
    <Laminator>
        <L1Laminate>Installed</L1Laminate>
        <L1LaminateType>33624064</L1LaminateType>
        <L1LaminateRemaining>65</L1LaminateRemaining>
        <L1LaminateSerialNumber>e0055000008e7624</L1LaminateSerialNumber>
        <L1LaminateLotCode>DPS 0913 </L1LaminateLotCode>
        <L1LaminatePartNumber>508668901</L1LaminatePartNumber>
        <L2Laminate>Installed</L2Laminate>
        <L2LaminateType>34078720</L2LaminateType>
```

```
        <L2LaminateRemaining>97</L2LaminateRemaining>
        <L2LaminateSerialNumber>e0055000008e6c08</L2LaminateSerialNumber>
        <L2LaminateLotCode>DPS 0913 </L2LaminateLotCode>
        <L2LaminatePartNumber>508832901</L2LaminatePartNumber>
    </Laminator>
    <TactileImpresser>
        <TactileImpresser>Installed</TactileImpresser>
        <TactileImpresserFoilType>1</TactileImpresserFoilType>
        <TactileImpresserFoilRemaining>99</TactileImpresserFoilRemaining>
        <TactileImpresserSerialNumber>E00550000BD68247</TactileImpresserSerialNumber>
        <TactileImpresserLotCode>10292019t</TactileImpresserLotCode>
        <TactileImpresserPartNumber>525067003</TactileImpresserPartNumber>
    </TactileImpresser>
</PrinterSupplies3>
```

Element Value	Description
Printer	
PrintRibbon	Indicates if a print ribbon is installed. The values are: • None • Installed
PrintRibbonType	The type of ribbon installed in the printer. The value returned is one of the supported ribbon types for the printer, such as YMCKT, ymcKT, KT, FCMYP-KP, and so on.
RibbonRemaining	The amount of unused ribbon as a percent.
RibbonSerialNumber	The serial number of the ribbon.
RibbonLotCode	The lot code of the ribbon.
RibbonPartNumber	The part number of the ribbon.
RibbonRegionCode	The region code of the ribbon.
RetransferFilmPartNumber	The part number of the retransfer film. Reported for the CR805 retransfer card printer only.
RetransferFilmPercent Remaining	The amount of unused retransfer film as a percent. Reported for the CR805 retransfer card printer only.
RetransferFilmSerialNumber	The serial number of the retransfer film. Reported for the CR805 retransfer card printer only.
RetransferFilmLotCode	The lot code of the retransfer film. Reported for the CR805 retransfer card printer only.
Embosser	

Element Value	Description
IndentRibbon	If the system includes an embosser, this element indicates if indent ribbon is installed. The values are: • None • Installed
IndentRibbonType	The type of indent ribbon installed.
IndentRibbonRemaining	The amount of unused indent ribbon as a percent.
IndentRibbonSerialNumber	The serial number of the indent ribbon.
IndentRibbonLotCode	The lot code of the indent ribbon.
IndentRibbonPartNumber	The part number of the indent ribbon.
TopperRibbon	This element indicates if topping foil is installed. The values are: • None • Installed
TopperRibbonType	The type of topping foil installed in the printer. The values are: • Silver • Gold • Black • White • Blue
TopperRibbonRemaining	The amount of unused topping foil as a percent.
TopperRibbonSerialNumber	The serial number of the topping foil.
TopperRibbonLotCode	The lot code of the topping foil.
TopperRibbonPartNumber	The part number of the topping foil.
Laminator	
L1Laminate	Indicates if the laminator L1 station has a supply installed. The values are: • None • Installed
L1LaminateType	The universal supply code of the supply.

Element Value	Description
L1LaminateRemaining	The amount of unused supply as a percent.
L1LaminateSerialNumber	The serial number of the supply.
L1LaminateLotCode	The lot code of the supply.
L1LaminatePartNumber	The part number of the supply.
L2Laminate	Indicates if the laminator L2 station has a supply installed. The values are: • None • Installed
L2LaminateType	The universal supply code of the supply.
L2LaminateRemaining	The amount of unused supply as a percent.
L2LaminateSerialNumber	The serial number of the supply.
L2LaminateLotCode	The lot code of the supply.
L2LaminatePartNumber	The part number of the supply.
Tactile Impression Module	
TactileImpresserFoilType	The type of tactile foil installed in the tactile impression module. The foil type is returned by the firmware. The following impresser foil types are available: • Silver Metallic • Gold Metallic • Copper Metallic • Black • White • Clear
TactileImpresserFoilRemaining	The amount of unused foil as a percent.
TactileImpresserSerialNumber	The serial number of the tactile foil.
TactileImpresserLotCode	The lot code of the tactile foil.
TactileImpresserPartNumber	The part number of the tactile foil.

Sample Code—Supplies Status

For working code that demonstrates supplies status, refer to the following samples:

Visual C++, Visual C#, and VB.NET	status
Java	printer_status

Card Counts

Your application can get the card count information and reset the printer's resettable card counts.

Get Card Counts

To get the card count information stored in the printer using the IBidiSpl interface, set the schema to `Printer.CounterStatus2:Read`. For Java, call the GetPrinterCounterStatus2 method of the Java helper DLL (dxp01sdk_IBidiSpl_interop.dll).

The request returns the supplies status XML file.

Status XML File for Single Input Hopper Printer

```
<?xml version="1.0"?>
<!--Printer counter2 xml file.-->
<CounterStatus2>
    <PrinterStatus>Ready</PrinterStatus>
    <CurrentPickedExceptionSlot>18</CurrentPickedExceptionSlot>
    <TotalPickedExceptionSlot>18</TotalPickedExceptionSlot>
    <CurrentPickedInputHopper1>328</CurrentPickedInputHopper1>
    <TotalPickedInputHopper1>328</TotalPickedInputHopper1>
    <CurrentPicked>346</CurrentPicked>
    <TotalPicked>346</TotalPicked>
    <CurrentCompleted>245</CurrentCompleted>
    <TotalCompleted>245</TotalCompleted>
    <CurrentRejected>84</CurrentRejected>
    <TotalRejected>84</TotalRejected>
    <CurrentLost>17</CurrentLost>
    <TotalLost>17</TotalLost>
    <CardsPickedSinceCleaningCard>226</CardsPickedSinceCleaningCard>
    <CleaningCardsRun>5</CleaningCardsRun>
</CounterStatus2>
```

Status XML for Six-Position Input Hopper Printer

```
<?xml version="1.0"?>
<!--Printer counter2 xml file.-->
<CounterStatus2>
    <PrinterStatus>Ready</PrinterStatus>
    <CurrentPickedExceptionSlot>3</CurrentPickedExceptionSlot>
    <TotalPickedExceptionSlot>3</TotalPickedExceptionSlot>
    <CurrentPickedInputHopper1>156</CurrentPickedInputHopper1>
    <TotalPickedInputHopper1>156</TotalPickedInputHopper1>
    <CurrentPickedInputHopper2>1</CurrentPickedInputHopper2>
    <TotalPickedInputHopper2>1</TotalPickedInputHopper2>
    <CurrentPickedInputHopper3>4</CurrentPickedInputHopper3>
    <TotalPickedInputHopper3>4</TotalPickedInputHopper3>
    <CurrentPickedInputHopper4>4</CurrentPickedInputHopper4>
    <TotalPickedInputHopper4>4</TotalPickedInputHopper4>
    <CurrentPickedInputHopper5>6</CurrentPickedInputHopper5>
    <TotalPickedInputHopper5>6</TotalPickedInputHopper5>
    <CurrentPickedInputHopper6>4</CurrentPickedInputHopper6>
    <TotalPickedInputHopper6>4</TotalPickedInputHopper6>
    <CurrentPicked>178</CurrentPicked>
    <TotalPicked>178</TotalPicked>
    <CurrentCompleted>170</CurrentCompleted>
    <TotalCompleted>170</TotalCompleted>
    <CurrentRejected>8</CurrentRejected>
    <TotalRejected>8</TotalRejected>
    <CurrentLost>0</CurrentLost>
    <TotalLost>0</TotalLost>
    <CardsPickedSinceCleaningCard>164</CardsPickedSinceCleaningCard>
    <CleaningCardsRun>1</CleaningCardsRun>
</CounterStatus2>
```

Element Value	Description
CurrentPickedExceptionSlot	Number of cards picked from the exception slot.
TotalPickedExceptionSlot	Total number of cards picked from the exception slot.
CurrentPickedInputHopper1– Current PickedInputHopper6	Number of cards picked from the input hopper. For a multi-hopper printer, the number of cards picked from each hopper is returned. This can be reset at the printer with proper permission.
TotalPickedInputHopper1– TotalPickedInputHopper6	Total number of cards picked from the input hopper. For a multi-hopper printer, the total number of cards picked from each hopper is returned.
CurrentPicked	Number of cards picked by the printer. This can be reset at the printer with proper permission.
TotalPicked	Total number of cards picked by the printer.

Element Value	Description
CurrentCompleted	Number of cards successfully completed by the printer. This can be reset at the printer with proper permission
TotalCompleted	Total number of cards successfully completed by the printer.
CurrentRejected	Number of cards that were rejected by the printer because they failed or were canceled. This can be reset at the printer with proper permission.
TotalRejected	Total number of cards that were rejected by the printer because they failed or were canceled.
CurrentLost	A calculated value for the cards that were neither completed nor rejected. This can be reset at the printer with proper permission.
TotalLost	Total number of cards that were neither completed nor rejected.
CardsPickedSinceCleaningCard	Number of cards the printer has picked since it was cleaned. This resets when the first card is picked after the printer has been cleaned.
CleaningCardsRun	Number of cleaning cards that have been run through the printer.

Reset Card Counts

To reset the resettable card count values stored in the printer using the IBidiSpl interface, set the schema to `Printer.ResetCardCount:Set`. For Java, call the ResetCardCounts method of the Java helper DLL (dxp01sdk_IBidiSpl_interop.dll).

Sample Code—Card Counts

For working code that demonstrates card counts, refer to the following samples:

Visual C++, Visual C#, and VB.NET	status—Use to obtain card count information printer_control—Use to reset card counts
Java	printer_status—Use to obtain card count information printer_control—Use to reset card counts

Hopper Status

Your application can get the status of the input hoppers (whether or not cards are present in the hopper) for the CR805 retransfer card printer.

Get Hopper Status

To get the hopper status for the printer using the IBidiSpl interface, set the schema to `Printer.Hopper:Status.Get`. For Java, call the GetHopperStatus method of the Java helper DLL (dxp01sdk_IBidiSpl_interop.dll).

The request returns the hopper status XML file.

Input Hopper Status XML File for a Retransfer Card Printer

```
<!-- For CR805 printer where hopper detection is available -->
<?xml version="1.0"?>
<HopperStatus>
    <PrinterStatus>Ready</PrinterStatus>
    <HopperDetection>Available</HopperDetection>
    <HopperInformation Name="Exception" Type="Input" Status="Cannot Detect"
      CardStock="CardStock1"/>
    <HopperInformation Name="Hopper1" Type="Input" Status="Cards Present"
      CardStock="CardStock1"/>
    <HopperInformation Name="Hopper2" Type="Input" Status="Cards Present"
      CardStock="CardStock1"/>
    <HopperInformation Name="Hopper3" Type="Input" Status="Empty"
      CardStock="CardStock1"/>
    <HopperInformation Name="Hopper4" Type="Input" Status="Empty"
      CardStock="CardStock1"/>
    <HopperInformation Name="Hopper5" Type="Input" Status="Empty"
      CardStock="CardStock1"/>
    <HopperInformation Name="Hopper6" Type="Input" Status="Empty"
      CardStock="CardStock1"/>
</HopperStatus>
```

The hopper status information retrieved from the printer depends on whether the printer has a single hopper or a multi-hopper.

- For a multi-hopper printer, the firmware cannot detect if a card is present in the exception hopper. The firmware always returns the status “Cannot detect.”
- For a single-hopper printer, the firmware can detect if a card is present in the exception hopper. The firmware returns the status “Cards Present” or “Empty.”

Input Hopper Status XML File for a Non-Retransfer Printer

The hopper status is not available for a non-retransfer printer. The following XML shows the information returned for a non-retransfer printer.

```
<!-- For non-retransfer printer where hopper cannot be detected. -->
<?xml version="1.0"?>
<HopperStatus>
    <PrinterStatus>Ready</PrinterStatus>
    <HopperDetection>Unavailable</HopperDetection>
    <HopperInformation Name="None" Type="" Status="" CardStock=""/>
    <HopperInformation Name="None" Type="" Status="" CardStock=""/>
    <HopperInformation Name="None" Type="" Status="" CardStock=""/>
    <HopperInformation Name="None" Type="" Status="" CardStock=""/>
    <HopperInformation Name="None" Type="" Status="" CardStock=""/>
    <HopperInformation Name="None" Type="" Status="" CardStock=""/>
</HopperStatus>
```

Locking

If your printer is equipped with locks, your application can lock and unlock the printer, as well as change the password needed to unlock the printer. The IBidiSpl requests used to do this are:

- Printer.Locks:ChangeLockState:Set
- Printer.Locks:ChangePassword:Set

Java uses the following functions with the Java helper DLL (dxp01sdk_IBidiSpl_interop.dll):

- SetPrinterLockState
- ChangeLockPassword

Lock or Unlock the Printer

Your application must create an XML structure with the lock state and password. The driver receives this XML formatted data as a BIDI_BLOB.

```
<?xml version="1.0"?>
<ChangeLocks>
    <LockPrinter>%d</LockPrinter>
    <CurrentPassword>%ls</CurrentPassword>
</ChangeLocks>
```

LockPrinter Value	Description
1	Lock printer
2	Unlock printer

The CurrentPassword value must be set to the correct password to successfully lock or unlock the printer.

Change the Lock/Unlock Password

Your application must create an XML structure with the lock state and password. The driver receives this XML formatted data as a BIDI_BLOB.

```
<?xml version="1.0"?>
<ChangeLocksPassword>
    <LockPrinter>1</LockPrinter>
    <CurrentPassword>test</CurrentPassword>
    <NextPassword>abcd</NextPassword>
</ChangeLocksPassword>
```

Your application must supply both the correct CurrentPassword and the new password in the NextPassword element.

`LockPrinter` is always set to 1. Changing the lock password locks the printer if it is unlocked.

Password Rules

Use the following rules to make sure that the password is considered valid by the printer:

- A password must have at least 4 legal characters. Legal characters are:
  - alphanumeric (English) (A–Z, a–z, 0–9)
  - plus (+)
  - slash (/)
  - dollar sign ($)
- A password is case sensitive.
- Empty quotes ("") are used to disable the locking password.

  If the printer is configured to not require a password, the printer locks or unlocks ignoring whatever password is sent.
- When the locking password is changed, the NextPassword value becomes the CurrentPassword for the next attempt to lock or unlock the printer.

  When you send empty quotes ("") as the NextPassword value, the printer no longer requires a password to lock or unlock.

Determine the Success of a Lock Request

For both lock requests, the status is returned in another XML structure. The following is an example of an attempt to lock a printer that does not have locks installed.

```
<?xml version="1.0"?>
<!--Printer status xml file.-->
<PrinterStatus>
    <ClientID>agarwas-Win7_{32DCD216-3B4E-4806-9661-3F80D6D99F72}</ClientID>
    <WindowsJobID>0</WindowsJobID>
    <PrinterJobID>0</PrinterJobID>
    <ErrorCode>511</ErrorCode>
    <ErrorSeverity>2</ErrorSeverity>
    <ErrorString>Message 511: Cannot lock or unlock the printer. Locks are not
    installed.</ErrorString>
    <DataFromPrinter><![CDATA[  ]]></DataFromPrinter>
</PrinterStatus>
```

Sample Code—Locking

For working code that demonstrates the lock operation, refer to the following samples:

Visual C++, Visual C#, and VB.NET	locks
Java	locks

Change Color Settings

- The color settings apply only to SD-, CD-, and CE-series direct-to-card printers running D3.17.4 or newer firmware, and Sigma direct-to-card printers running D4 firmware.
- We recommend that you become familiar with color theory before adjusting color values.

The color adjust settings consist of 33 values for three color arrays: Red, Green, and Blue. Each array has 11 values that you can set to customize the color printing on a card. Your application can adjust how colors are printed by the printer.

Each color contains 11 comma-separated integers in a range from -25 to 25. These adjust the color settings to a value between -10% and +10% of the default setting.

Only the following characters are allowed when entering values:

- Plus (+)
- Minus (-)
- Comma (,)
- 0 through 9

Keep the following in mind when entering the values:

- Negative values must start with a minus (-). Positive values may start with a plus (+) or have no sign.
- The value zero cannot be preceded by a sign character, so plus or minus zero (for example, -0 or +0) is not allowed.
- Spaces are not allowed.
- A value that is not valid, such as a format error or an out-of-range value, causes a job error, and no changes are made to the settings.

Change the Color Values

To change the color values, your application calls the IBidiSpl interface with the schema set to `Printer.AdjustColor:Set`. For Java, call the SetColorAdjust method of the Java helper DLL (dxp01sdk_IBidiSpl_interop.dll).

It sends an XML structure containing all 33 settings as described in the ADJUST_COLOR_XML.

```
?xml version="1.0"?>
<AdjustColor>
    <RedColorChannel>%ls</RedColorChannel>
    <GreenColorChannel>%ls</GreenColorChannel>
    <BlueColorChannel>%ls</BlueColorChannel>
</AdjustColor>
```

Change One Color Channel

The following example changes the settings for the Red color channel, while leaving the Green and Blue color channels unchanged.

```
<?xml version="1.0"?>
<AdjustColor>
    <RedColorChannel>1,2,-1,0,25,6,-19,12,13,14,15</RedColorChannel>
    <GreenColorChannel></GreenColorChannel>
    <BlueColorChannel></BlueColorChannel>
</AdjustColor>
```

Change Two Color Channels

The following example changes the Green and Red color channels, while leaving the Blue color channel unchanged.

```
<?xml version="1.0"?>
<AdjustColor>
    <RedColorChannel>1,2,-1,0,25,6,-19,12,13,14,15</RedColorChannel>
    <GreenColorChannel>0,10,5,0,-7,-10,-5,0,0,0,0</GreenColorChannel>
    <BlueColorChannel></BlueColorChannel>
</AdjustColor>
```

Set the Color Values to Default Settings

To set one or more color channels back to their default settings, your application calls the IBidiSpl interface with the schema set to `Printer.SetDefaultColor:Set.` For Java, call the DefaultColorAdjust method of the Java helper DLL (dxp01sdk_IBidiSpl_interop.dll).

It sends an XML structure that specifies which color channels should be set to default as described in the DEFAULT_COLOR_XML.

```
<?xml version="1.0"?>
<SetDefaultColor>
    <RedColorChannel>%ls</RedColorChannel>
    <GreenColorChannel>%ls</GreenColorChannel>
    <BlueColorChannel>%ls</BlueColorChannel>
</SetDefaultColor>
```

Set All Color Channels to Default

The following example returns all color channels to their default values.

```
<?xml version="1.0"?>
<SetDefaultColor>
    <RedColorChannel>true</RedColorChannel>
    <GreenColorChannel>true</GreenColorChannel>
    <BlueColorChannel>true</BlueColorChannel>
</SetDefaultColor>
```

Set Two Color Channels to Default

The following example returns the Red and Green color channels to their default values, but leaves the Blue color channel unchanged.

```
<?xml version="1.0"?>
<SetDefaultColor>
    <RedColorChannel>true</RedColorChannel>
    <GreenColorChannel>true</GreenColorChannel>
    <BlueColorChannel>false</BlueColorChannel>
</SetDefaultColor>
```

Sample Code—Color Adjust

For working code that demonstrates the color adjust operation, refer to the following samples:

Visual C++, Visual C#, and VB.NET	printer_control
Java	printer_control

Change Color Values in Printer Manager or Printer Dashboard

The color adjust settings also are available through the Printer Manager web interface and the Printer Dashboard.

1. Log on to Printer Manager at the **WebService** access level, or sign on to the Printer Dashboard as **Service**.
2. In Printer Manager, select **Printer Setting > Print**. In the Printer Dashboard, select **Configuration > Settings >Print**.
3. Enter the values for the Color Adjust settings you want to change:
   - ColorAdjustB[0–10]
   - ColorAdjustG[0–10]
   - ColorAdjustR[0–10]

   The allowed range for each setting is [-25 – 25].
4. In Printer Manager, click **Set Current** to save your changes. In the Printer Dashboard, click **Save**.

Activate or Deactivate the Printer

Your application can activate or deactivate the printer when it will be out of service and you want to guarantee that it is not used to produce cards. The printer can be activated later to re-enable it for card production. The IBidiSpl request used to do this is `Printer.ActivatePrinter:Set`. For Java, call the ActivateOrDisablePrinter method of the Java helper DLL (dxp01sdk_IBidiSpl_interop.dll).

Your application must create an XML structure with the activation state and password. The driver receives this XML formatted data as a BIDI_BLOB.

```
<?xml version="1.0"?>
<ActivatePrinter>
      <Activate>%d</Activate>
      <Password>%ls</Password>
</ActivatePrinter>
```

Activate Value	Description
1	Activate printer
0	Deactivate printer

- To deactivate the printer, the password value must be set to the valid password.
- To activate the printer, the password value must match the password that was used to deactivate the printer.

  Caution: If the password used to deactivate the printer is lost, the printer cannot be returned to the activated state. The printer must be replaced.

- The password is reset when the printer is activated. This allows the application to use any valid password to deactivate the printer in the future.
- When a deactivate command is successful, the printer's front panel LCD displays **Print Job queue is deactivated**. A driver SDK status action returns the PrinterStatus as **unavailable**.
- If the command contains an invalid password, the SDK returns a 500 in the ErrorCode element of the returned PrinterStatus structure.

Refer to "Locking" on page 69 for more information about setting or changing the password.

Sample Code—Activate or Deactivate Printer

For working code that demonstrates the activate or deactivate operation, refer to the following samples:

Visual C++, Visual C#, and VB.NET	locks
Java	locks

Change the Printer State

Your application can change the printer state to Online, Offline, or Suspended. An application can print only when the printer state is Online. You can run maintenance operations when the printer state is Suspended. A printer that is Offline cannot be used. The IBidiSpl request used to do this is: `Printer.ChangePrinterState:Set`. For Java, call the ChangePrinterState method of the Java helper DLL (dxp01sdk_IBidiSpl_interop.dll).

Your application must create an XML structure with the printer state. The driver receives this XML formatted data as a BIDI_BLOB.

```
<?xml version="1.0"?>
<ChangePrinterState>
      <State>%d</State>
</ChangePrinterState>
```

State Value	Description
0	Online
1	Suspended
2	Offline

Sample Code—Change the Printer State

For working code that demonstrates the printer state operation, refer to the following samples:

Visual C++, Visual C#, and VB.NET	printer_state
Java	printer_state

Restart the Printer

Your application can restart a printer using the IBidiSpl interface with the schema set to `Printer.Restart:Set`. For Java, call the RestartPrinter method of the Java helper DLL (dxp01sdk_IBidiSpl_interop.dll).

Sample Code—Restart Printer

For working code that demonstrates a printer restart, refer to the following samples:

Visual C++, Visual C#, and VB.NET	printer_control
Java	printer_control

Shut Down the Printer

Your application can shut down a printer using the IBidiSpl interface with the schema set to `Printer.PowerDown:Set`. Printer shutdown is not supported by Java.

Sample Code—Shut Down Printer

For working code that demonstrates a printer shutdown, refer to the following samples:

Visual C++, Visual C#, and VB.NET	printer_control
Java	Not available in Java.

Interactive Mode Best Practices

- When you use interactive mode operations for card personalization, the driver does not accept a job until the interactive operations for the active job complete. Your application must be able to manage the card production queue and retry a job if the job request is denied because another job is active. Refer to "Start and End an Interactive Job" on page 27.
- Your application should always verify that the printer is online before starting a job. Refer to "Installed Printer Status, Supplies Status, and Counter Status" on page 55 for information about how to request and interpret the printer status to determine if the printer is online.
- Your application should always check the Printer Status returned by an IBidiSpl request to determine if the request succeeded or failed.
- When recovering from an error while in interactive mode, always use the PrinterJobID value returned by the Start Job request. The currently active job in the printer is canceled if your application sends a cancel action with a printer job ID of 0. Unless this printer is dedicated to your application, the currently active job may not be the job you intend to cancel.

Appendix A: Error Description Strings

A

Message	Description
100	Request not supported
101	Job could not complete
102	Card not in position
103	Printer problem
104	Critical problem
105	Magstripe data error
106	Magstripe data not found
107	Magstripe read data error
108	Magstripe read no data
109	Print ribbon problem
110	Print ribbon out or missing
111	Card not picked
112	Card hopper empty
113	Close cover to continue
114	Cover opened during job
116	Magstripe not available
117	Reader not available
118	Print ribbon type problem

Message	Description
119	Print ribbon not supported
120	User paused the printer
121	Print ribbon not identified
122	Magstripe format problem
123	Insert new card side 1 up
124	Insert same card side 2 up
125	Emboss critical error
126	Emboss format error
127	Emboss transport error
128	Embosser card jam
129	Embosser topper jam
130	Embosser card entry jam
131	Embosser card exit jam
132	Embosser card stack full
133	Embosser card reject full
135	Indent ribbon supplies out
136	Indent ribbon break
137	Embosser wheel error
138	Embosser indent error
139	Lost card, open emboss cover
140	Embosser not available
141	Close emboss cover
142	Emboss cover error
143	Topping foil problem
144	Topping foil out

Message	Description
145	Topping foil type problem
146	Topping foil support err
147	Topping foil no tag found
149	Option not installed
150	Print while unlocked
151	Failed to lock
152	Insert new card side 2 up
153	Insert same card side 2 up
166	C2 supply out or missing
167	C2 supply not identified
168	C2 supply not supported
170	Insert new card side 1 up
171	Insert same card side 1 up
172	Insert cleaning card
173	Improper shutdown
175	C2 supply error
176	C2 supply type error
177	Laminator not available
196	Laminator error critical
197	Laminator entry card problem
198	L1 area card problem
199	L2 area card problem
200	Laminator exit card problem
201	L1 supply problem
202	L1 supply out or missing

Message	Description
203	L1 supply type problem
204	L1 supply not supported
205	L1 supply not identified
206	L2 supply problem
207	L2 supply out or missing
208	L2 supply type problem
209	L2 supply not supported
210	L2 supply not identified
211	L1 heater problem
212	L2 heater problem
213	L1 heater sensor problem
214	L2 heater sensor problem
215	L1 heater roller problem
216	L2 heater roller problem
217	Debow problem
218	Impresser problem
219	Impresser sensor problem
220	Impresser heater problem
221	Bar code scanner problem
222	Firmware version mismatch
223	Laminator system mismatch
224	Supply region not valid
225	Rewrite config mismatch
227	C2 waiting for roller temp
228	C1 printhead error

Message	Description
229	C2 heat sensor error
230	C2 heater error
231	C2 heated roller motion error
232	Retransfer debow error
233	Smart card contact fail
234	K1 transport card jam
235	K2 transport card jam
236	Flipper module rotate error
237	C1 supply out or missing
238	C1 supply not identified
239	C1 supply not supported
240	C1 supply error
241	C1 supply type error
242	RT1 reject tray full
243	Card lost
245	Configuration error
250	Card indexer error
251	Laser position error
252	Laser template error
253	Laser horizontal data error
254	Laser not available
255	Laser cover open
256	Laser interlock open
257	Laser air filter missing
258	Laser not detected

Message	Description
259	Laser firmware error
260	Laser control error
261	Laser option not supported
262	Vision operation failed
263	Laser power board error
264	Lost card in laser
265	Laser card entry jam
266	Laser card exit jam
267	Laser hardware failure
268	Laser setup error
269	Laser setup name length error
270	Laser text length error
271	Laser card output full
272	Laser card reject full
273	Laser output area error
274	Vision not available
275	Multi-hopper error
276	Multi-hopper card jam
277	Multi-hopper pick error
278	Insert cleaning card in H2
279	Card stock not compatible
280	TIM lost card
281	TIM heater error
282	TIM supply error
283	TIM foil error

Message	Description
284	TIM drive error
285	TIM supply read error
286	TIM supply write error
288	TIM not detected
289	TIM error
291	TIM foil not supported
292	TIM card jam
293	TIM door open
294	Cleaning Required
295	Settings error
297	Option Tag Error
298	Option Tag Success
500	The printer is not available
501	The printer connection was lost
502	The card data is missing or is not usable
504	The card data is missing or is not usable
505	USB communication issue
506	A card is currently processing
507	The printer is unlocked
508	The printer is shutting down
509	The printer is offline or suspended
510	The printer is unlocked
511	Cannot lock or unlock the printer. Locks are not installed.
512	Cannot lock or unlock the printer. The password is incorrect or invalid.

Message	Description
513	Cannot lock or unlock the printer. The printer is busy.
514	Cannot lock or unlock the printer. The cover is open.
515	Failed to lock or unlock the printer. The locks did not function.
516	Timeout expired before bar code could be read.
517	Wrong printer job ID.
518	Unable to print.
519	File import failed.
520	Hopper number not valid.
521	Start job failed.
522	Failed to read laser files.
523	Apply topcoat or overlay to the card

Appendix B: Use Eclipse to Create Java Samples

B

The XPS Driver SDK Java samples work with either the 32- or 64-bit Java runtimes. Before you can run the sample code that is in the **samples** folder, you first must create a common_java.jar file and then generate a runnable JAR file for each sample that you want to use. This chapter describes how to use the Eclipse development environment to generate the JAR files.

The examples in this chapter were created using Eclipse Version: Oxygen.31 Release (4.7.3a) and Java version 1.8.

You also can use the runnable JAR files that are included in the SDJ jars folder. Refer to Appendix C: "Use the SDK Java Samples" for complete information.

Extract the SDK Files

1. Make sure a Java runtime is installed on your computer.

   Open a command prompt window and Issue the command “java -version” from a command line. The following example shows the type of information that displays:

   ```
   C:\Users\username>java -version

   java version "1.8.0_181"
   Java(TM) SE Runtime Environment (build 1.8.0_181-b13)
   Java HotSpot(TM) 64-Bit Server VM (build 25.181=b13, mixed mode)
   ```

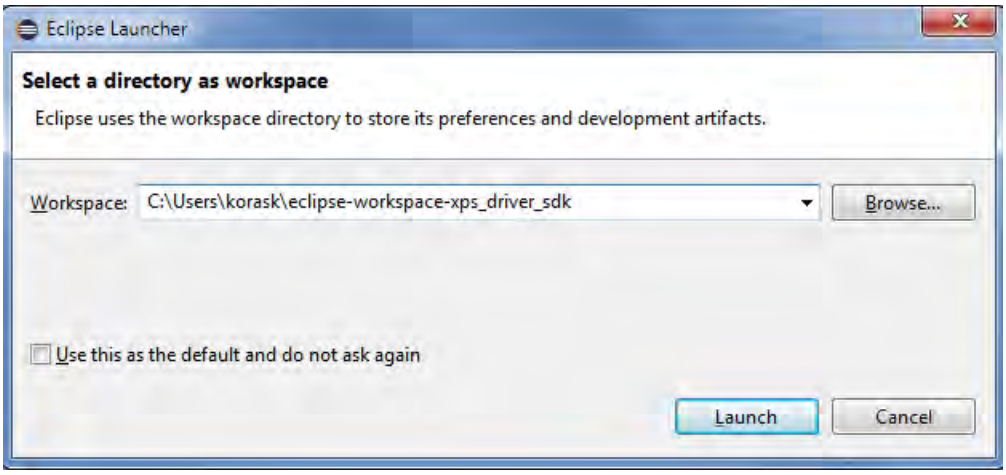
2. Download the XPS_Card_Printer_SDK.zip file to a folder. For example: C:\java\XPS_Card_Printer_SDK.

3. Extract the XPS Driver SDK zip file to the folder. The folder contains the following files:

   ```
   C:\java\xps_card_printer_sdk>dir

   Directory of C:\java\xps_card_printer_sdk

   07/16/2018  01:45 PM    <DIR>          exes
   06/12/2018  04:05 PM           488,447 Readme.pdf
   07/16/2018  01:45 PM    <DIR>          samples
   06/06/2018  09:26 AM         1,362,250 SDK_Dev_Guide.pdf
   07/06/2018  03:51 PM        17,103,709 XPS_Card_Printer_SDK.zip
   ```

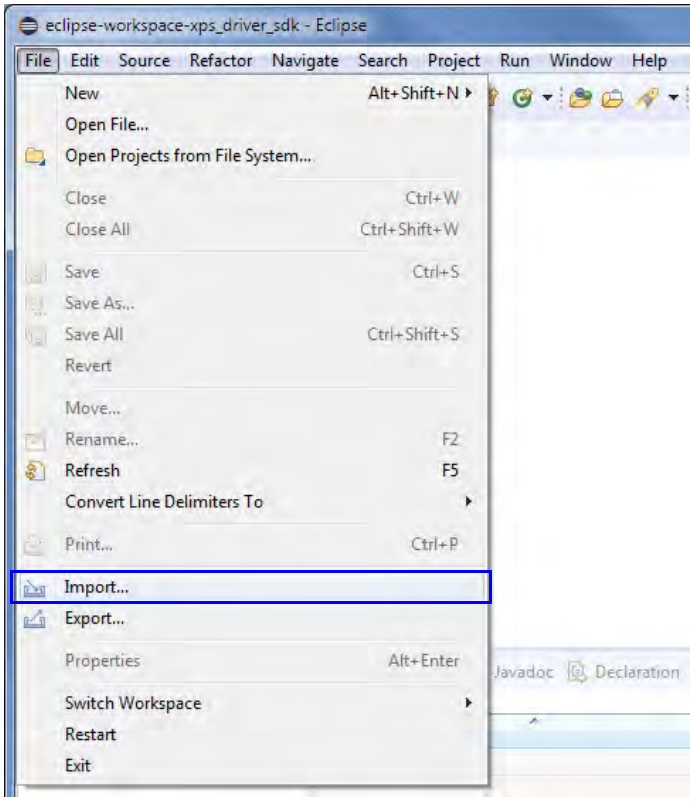
Create an Eclipse Workspace

1. Start Eclipse and create a new workspace.

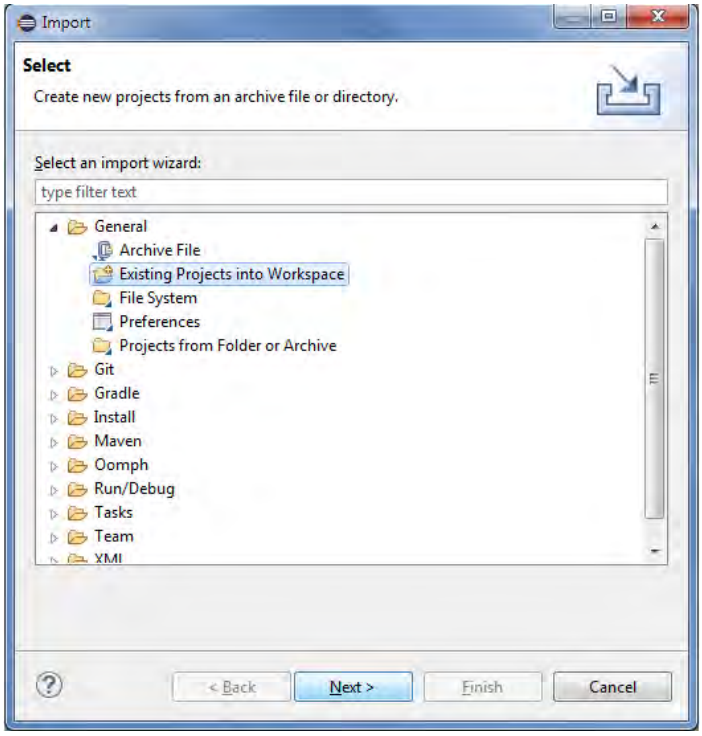


2. Import the SDK samples.

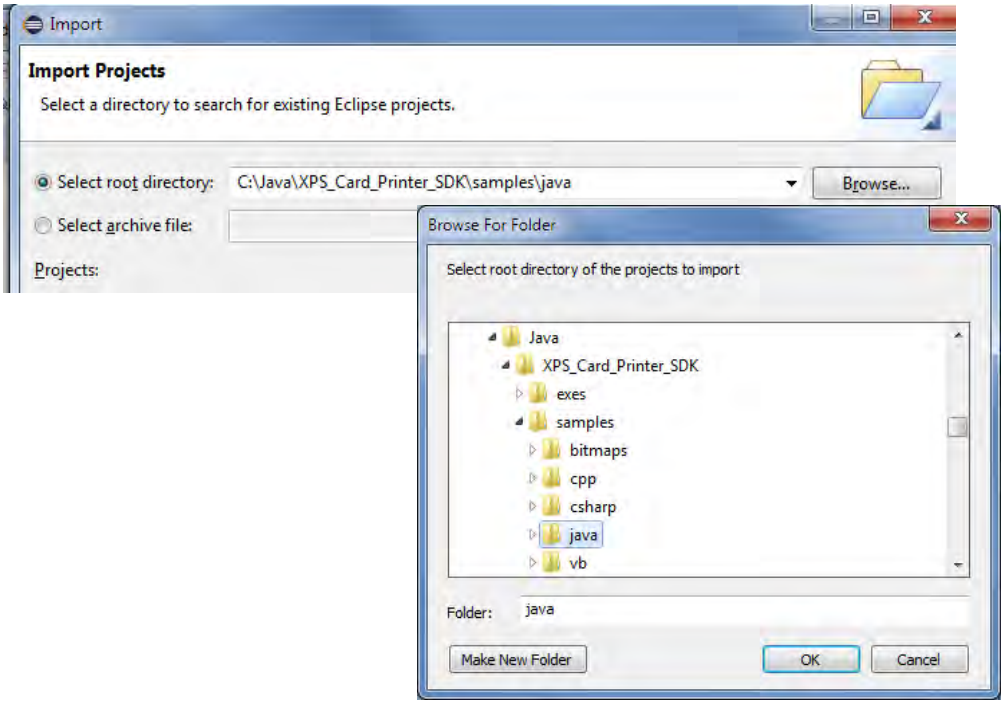
   a. When Eclipse opens, select **File > Import**.



b. In the Import window, select **General > Existing Projects into Workspace** and click **Next**.

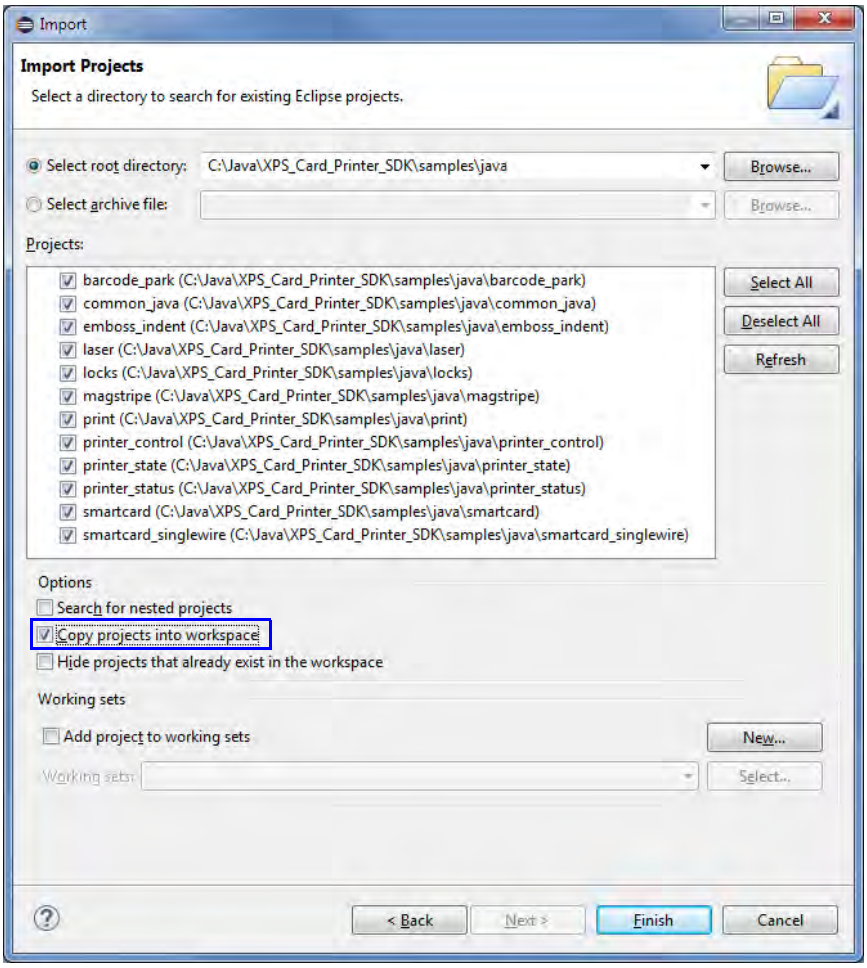


c. Browse to the **samples\java** folder under the folder you created in step 2 of "Extract the SDK Files" on page B-1 and click **OK**.



d. The Java sample files display in the Projects list. All the files already should be selected. If they are not, click **Select All**.

Select **Copy projects to workspace** as some samples have file location dependencies.

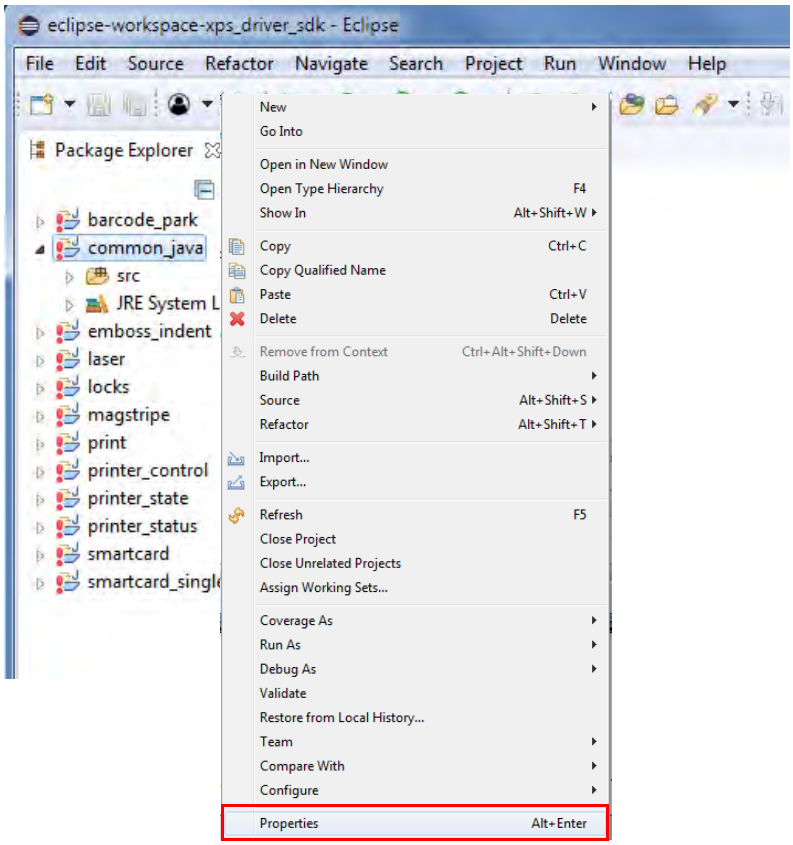


e. Click **Finish**.

Build the common_java JAR File

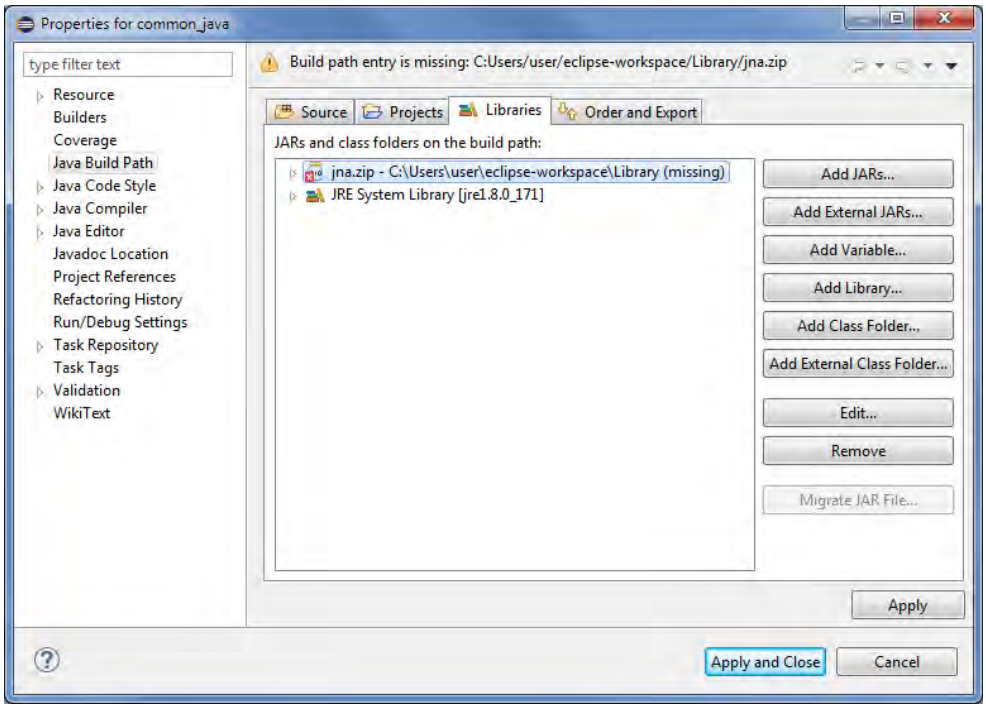
You must build a common_java package before running the Java samples. The common_java package is imported into each sample.

1. In the Eclipse Package Explorer, right-click the **common_java** project folder and select **Properties**.

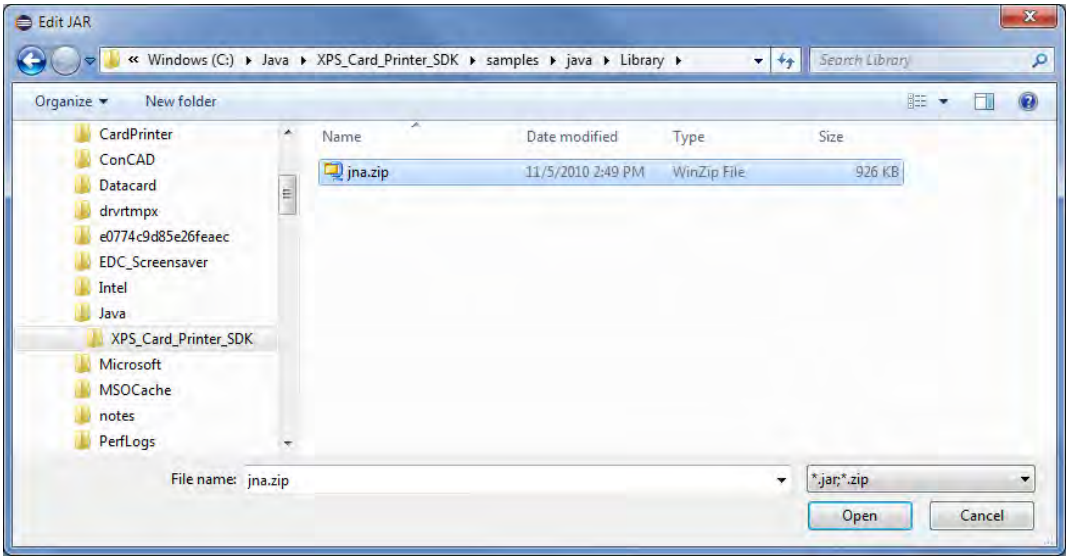


2. In the Properties window, select **Java Build Path** and click the **Libraries** tab.

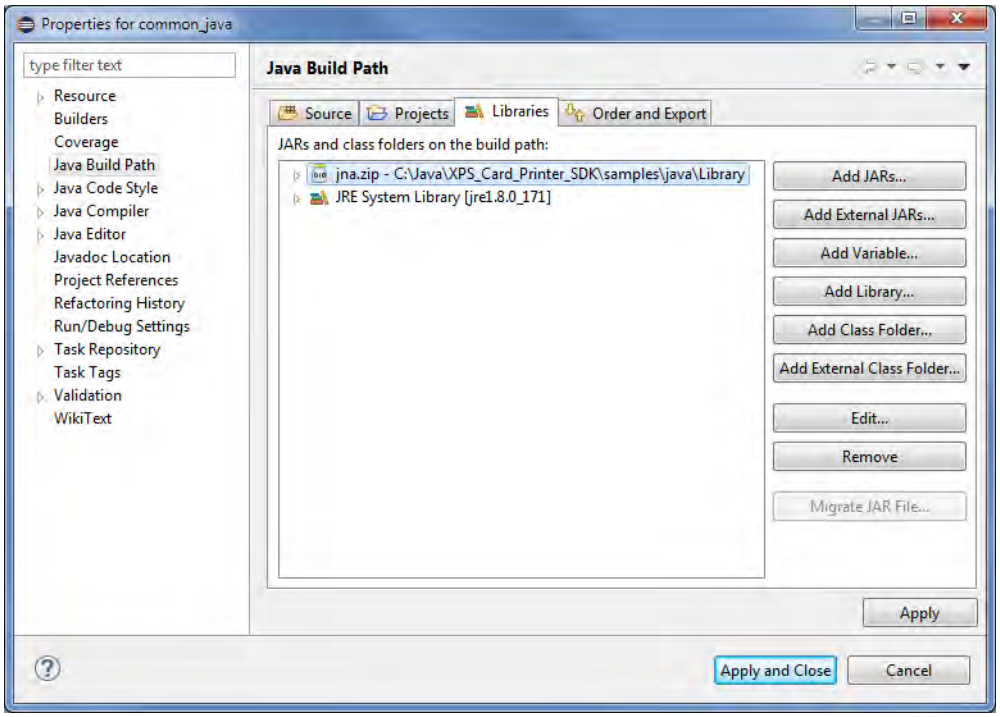
   Double-click the **jna.zip** entry to update the jna.zip file location.



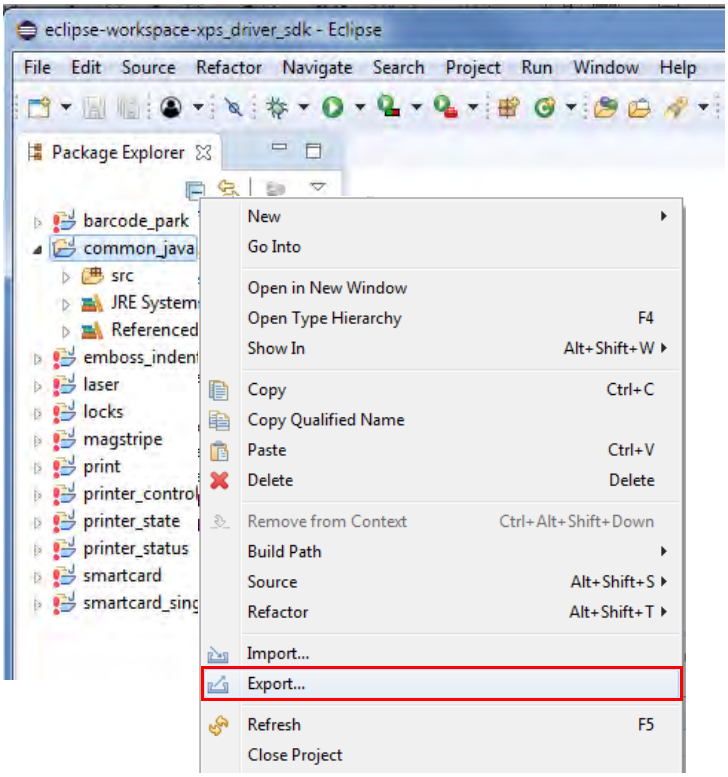
3. Locate the jna.zip file in the C:\Java\XPS_Card_Printer_SDK\samples\java\Library folder. Select the file and click **Open**.



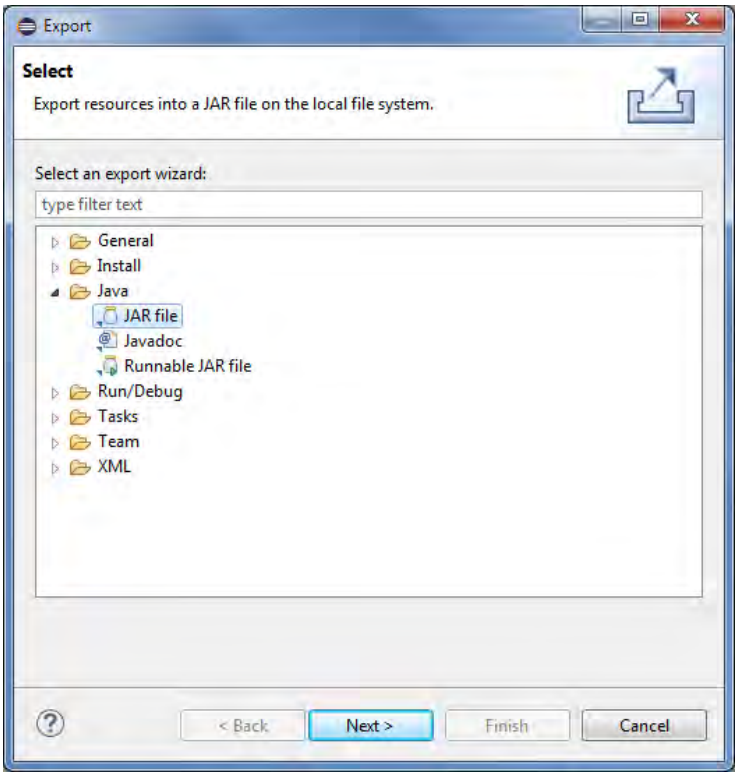
4. Click **Apply and Close** in the Properties window.



5. In the Eclipse Package Explorer, right-click the **common_java** project folder and select **Export**.



6. In the Export window, select **Java > JAR file** and click **Next**.



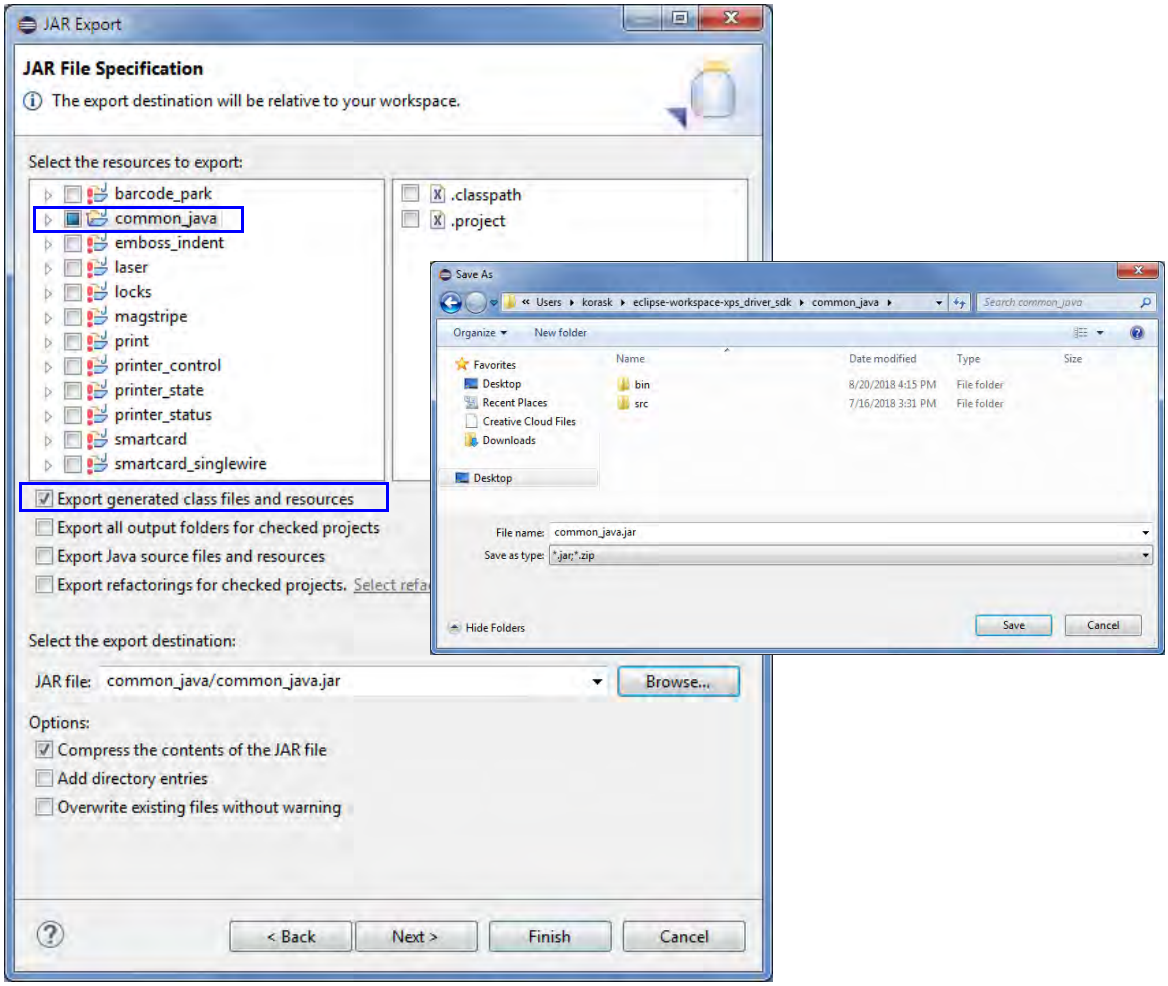
7. In the JAR Export window, do the following:

   a. Make sure that **common_java** is the only item selected in the resources pane.

   b. Clear the **.classpath** and **.project** check boxes.

   c. Make sure that **Export generated class files and resources** is selected.

   d. Select the export destination. Browse to the eclipse-workspace you created in step 1 of "Create an Eclipse Workspace" on page B-2. Select the **common_java** project folder and name the JAR file **common_java.jar**. Click **Save**.



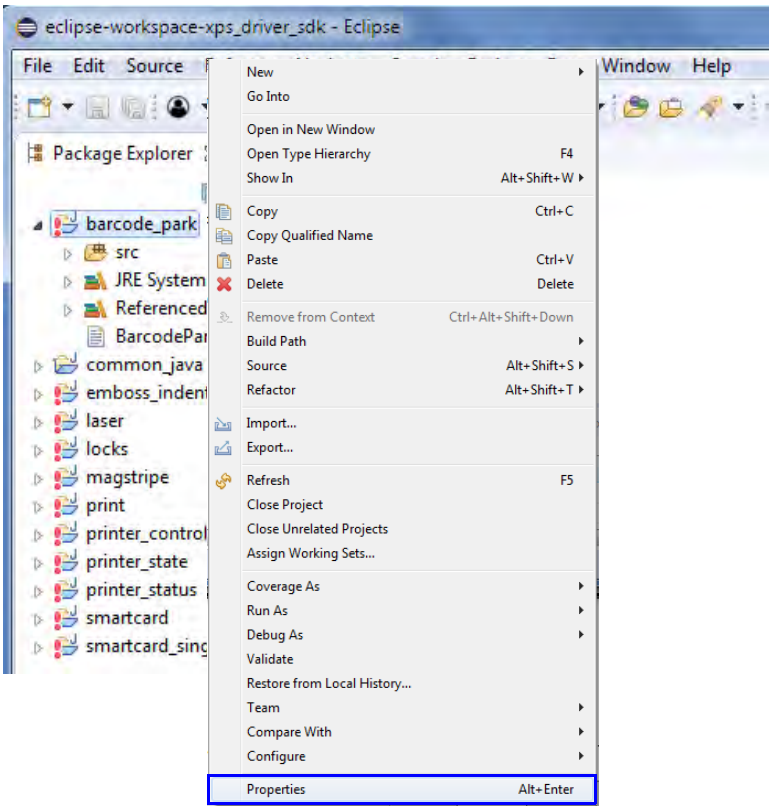
8. Click **Finish**.

Create Runnable JAR Files for Each Java Sample

After you build the common_java.jar file, you can create runnable JAR files for each Java sample that you want to use.

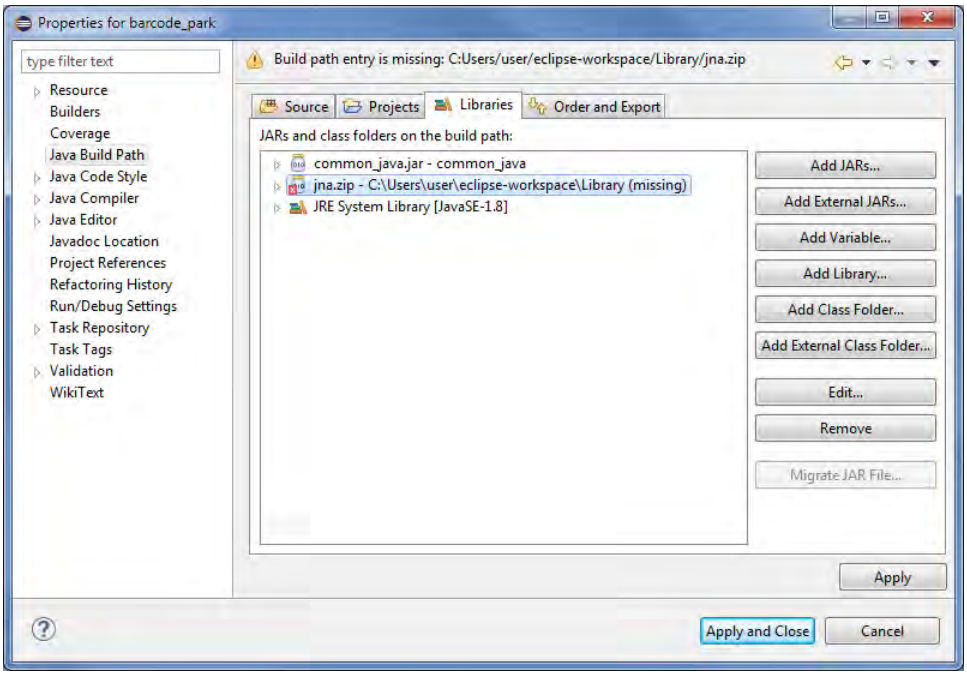
The following example illustrates creating a runnable JAR file for the barcode_park sample. Use the same procedure for each Java sample.

1. In the Eclipse Package Explorer, right-click the project folder of the sample you want to run (in this example, **barcode_park**) and select **Properties**.

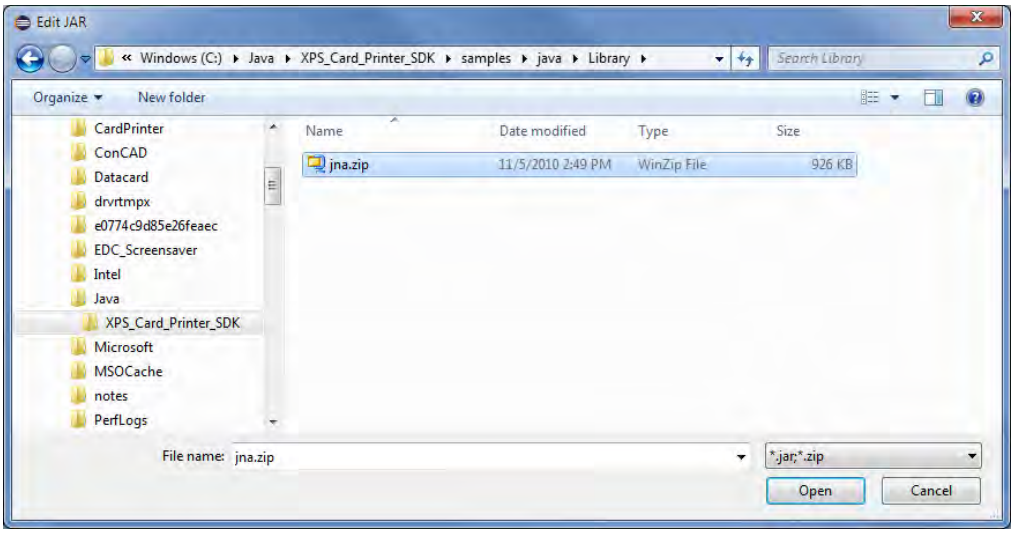


2. In the Properties window, select **Java Build Path** and click the **Libraries** tab.

   Double-click the **jna.zip** entry to update the jna.zip file location.

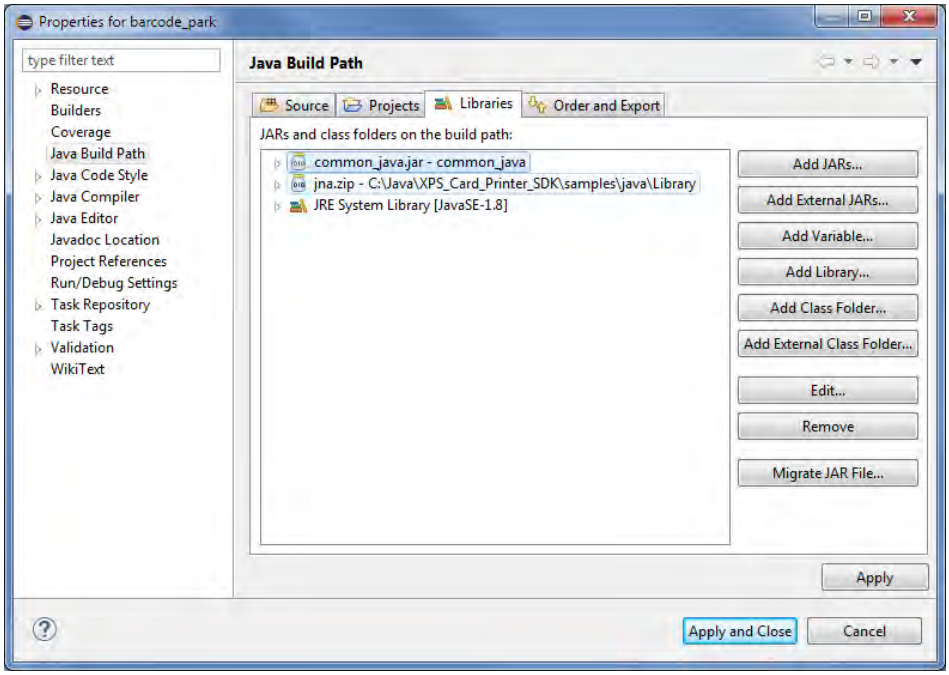


3. Locate the **jna.zip** file in the **C:\Java\XPS_Card_Printer_SDK\samples\java\Library** folder. Select the file and click **Open**.



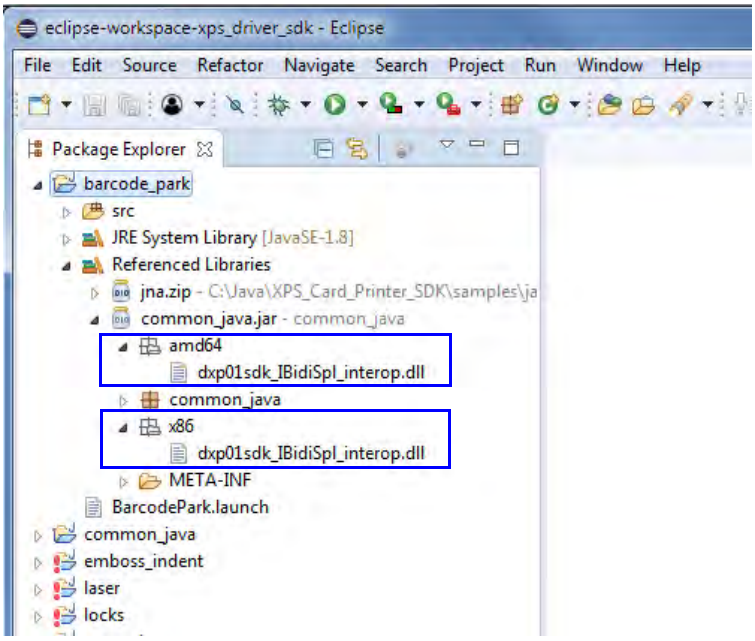
4. In the Properties window, the **common_java.jar** should point to the correct location.

   If it indicates that the location is missing, double-click the file entry and locate the common_java.jar file you created in "Build the common_java JAR File" on page B-5.



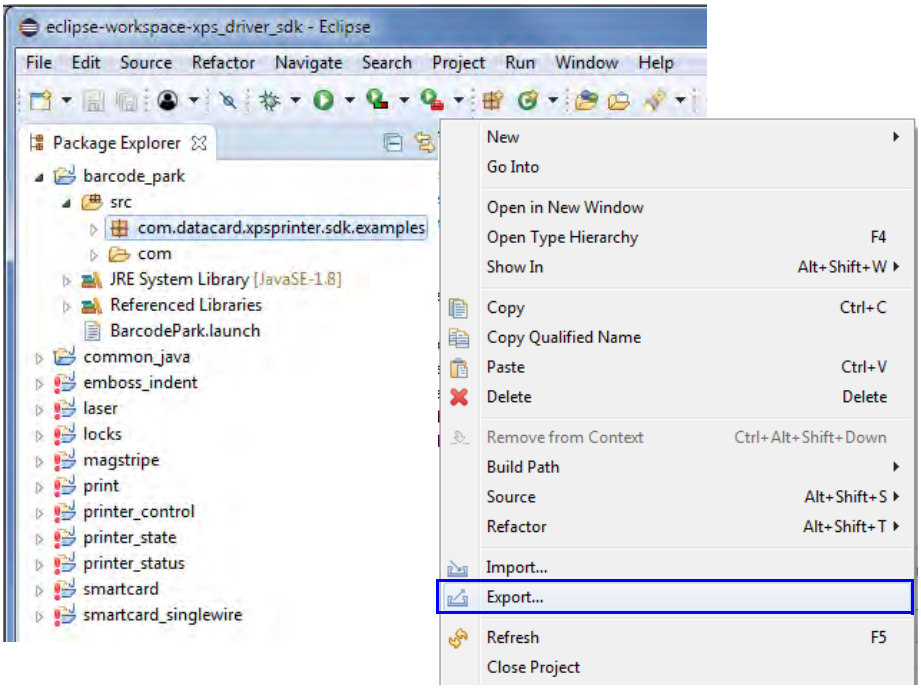
5. Click **Apply and Close** in the Properties window.

6. In the Package Explorer, select **barcode_park > Referenced Libraries > common_java.jar > amd64** and **x86**.

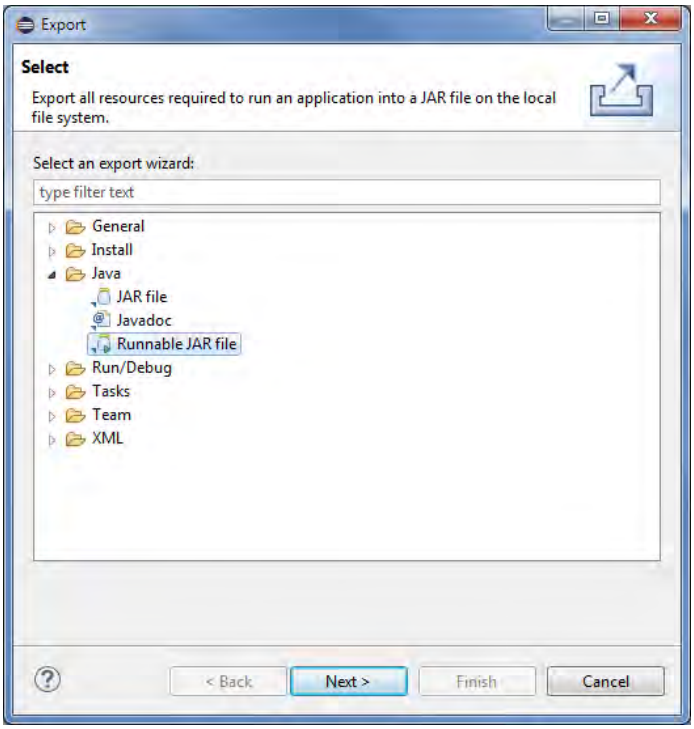


Verify that **dxp01sdk_IBidiSpl_interop.dll** displays for both 32-bit and 64-bit systems.

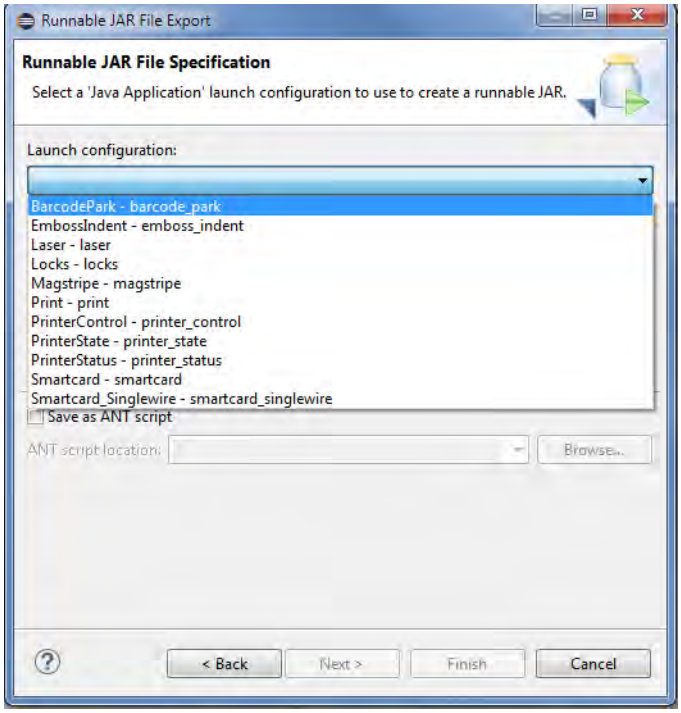
7. Select **barcode_park > src.** Right-click the project package, **com.datacard.xpsprinter.sdk.examples** and click **Export**.



8. In the Export window, select **Java > Runnable JAR file** and click **Next**.



9. Select the Launch configuration that corresponds to your project.

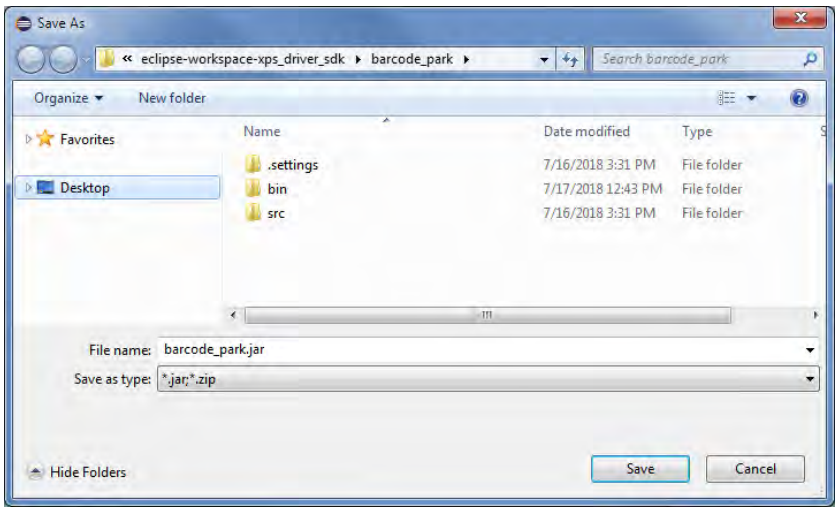


10. Specify the Export destination.

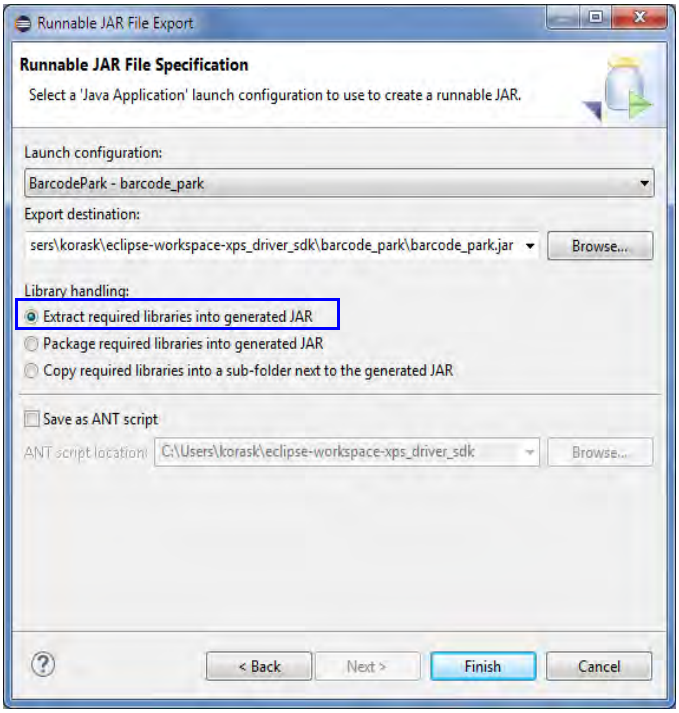
    a. Browse to the Eclipse workspace you created in "Create an Eclipse Workspace" on page B-2 and select **barcode_park**.

    The Eclipse workspace is the required destination for all Java samples.

    b. Enter the name of the sample file, **barcode_park.jar**, and click **Save**.

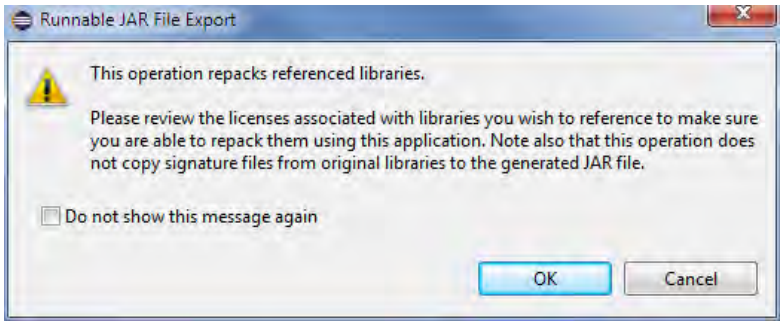


11. In the Runnable JAR file Export window, make sure that **Extract required libraries into generated JAR** is selected.



12. Click **Finish**.

    If the referenced library warning displays, click **OK**.



13. Repeat this process for each Java sample that you want to use.

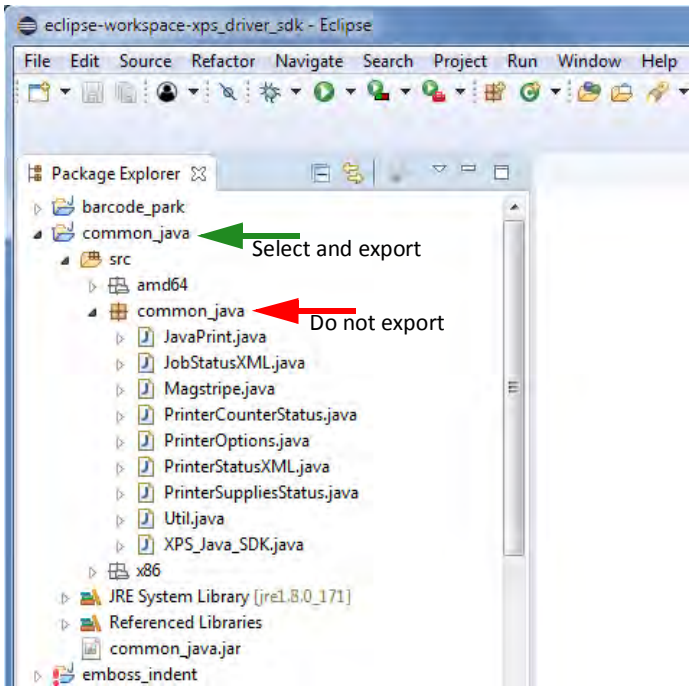
Run the JAR File

You can run the JAR file to see help information and the command line options.

Open a command prompt window in the location of the JAR file in the Eclipse workspace and enter the command `java -jar barcode_park.jar`.

```
C:\Users\korask\eclipse-workspace-xps_driver_sdk\barcode_park>java -jar barcode_
park.jar

barcode_park.jar demonstrates interactive mode parking a card in the barcode
reader, moving the card from the station, and options to print and poll
for job completion. Uses hardcoded data for printing.

java -jar barcode_park.jar -n <printername> [options]

options:
 -n <printername>. Required. Try -n "XPS card printer"
 -b parks the card such that the barcode on the back side of the card
    can be read. Default operation is to park the card such that the
    barcode on the front side of the card can be read.
 -c poll for job completion
 -p Print sample text on the card.
 -f <Front | Back>. Flip card on output
 -i <input hopper>. Defaults to input hopper #1.

Examples:
  java -jar barcode_park.jar -n “XPS Card Printer” -p

Parks a card such that the barcode on the front side of the card can be read,
asks you to continue (i.e., barcode read was successful) or reject (i.e. barcode
read was not successful). If the read was successful, sample images are printed
on one or both sides of the card.

  java -jar barcode_park.jar -n “XPS Card Printer” -b

Parks a card such that the barcode on the front side of the card can be read,
asks you to continue or reject and then does what you requested.
```

Troubleshooting

Error	Solution
Unable to find dxp01sdk_IBidiSpl_interop.dll	Make sure that you exported the common_java jar file at the project folder level instead of at the common_java package level. Refer to "Build the common_java JAR File" on page B-5. 

Recommendations

Use the following recommendations when creating or working with the Java samples.

- Create a new Eclipse workspace before you make changes to the Java samples. This maintains the original samples in their own workspace in case you need them.
- Create a new Eclipse workspace whenever you install a new version of the SDK. The helper dll, dxp01sdk_IBidiSpl_interop.dll, is copied to each sample in the workspace. When you create a new workspace it ensures that the dll is always up-to-date.

  You must rebuild your Java samples when you install a new version of the SDK.

- The helper dll, dxp01sdk_IBidiSpl_interop.dll, is backward compatible with previous versions of the SDK. However, because the method used to build the Java samples has changed, make sure that you create a new Eclipse workspace to use the current version of the dll.

Appendix C: Use the SDK Java Samples

C

Overview

The SDK Java files that are included in the jars folder are runnable JAR files that you can use without having to build the files yourself. This is similar to the compiled samples for Visual C++, Visual C#, and VB.NET located in the exes folder.

The Java samples were created and tested using Java version 8, update 181. We recommend that you install and use that version with the samples.

- You need to know the location where the Java version 8, update 181 java.exe file is installed on your computer to run the JAR files. java.exe is located in the bin folder.
  For example: C:\Program Files\Java\jre1.8.0_181\bin.
- If you run the Java samples with a later version of Java, such as Java version 9 or newer, Java may issue a warning message indicating that "an illegal reflective access operation has occurred." This does not affect the Java samples and they run without problems.

The Java SDK JAR files work with either the 32- or 64-bit versions of Java 8, update 181.

Use the Java Samples

The Java sample .jar files are placed in the **jars** folder on your computer when you extract the SDK zip file (refer to "Installation" on page 3). The common_java.jar file is included in the folder and is used by all the other samples. In addition, a separate laser folder in the jars folder contains the files used for engraving on a desktop laser system.

Do the following to run a JAR file:

1. Open a command window and point to the **jars** directory on your computer. For example,

   C:\XPS Driver_SDK\jars>

2. Enter the command to run the JAR file. You must point to the java.exe file in the Java bin directory on the computer. Then, enter the name of the JAR file you want to run.

   For example, enter the following command to run the barcode_park.jar sample to see help information and the command line options:

   C:\XPS Driver_SDK\jars>C:\Progra~1\Java\jre1.8.0_181\bin\java.exe -jar barcode_park.jar

   This example assumes that the java.exe file is in C:\Program Files\Java on a Windows 64-bit system. The command uses the short name for the Program Files directory (Progra~1).
   The location of the java.exe file can vary depending on where Java is installed on your computer or if you use a 32-bit Windows system.

After you run a Java sample, the Java helper dll file, dxp01sdk_IBidiSpl_interop.dll, is placed in the jars folder. The dxp01sdk_IBidiSpl_interop.dll file is used by the sample .jar files to communicate with the Card Printer Driver. The file is included in the common_java.jar file and is extracted to the jars folder when any of the sample .jar files is executed.

Appendix D: Suppress the Driver Message Display

D

If you want your application to present printer and driver messages to the user and resolve errors directly, you can suppress the display of messages by the driver. This is known as “silent mode.”

Enable Driver Silent Mode

1. Silent mode is enabled when the following registry setting is present and the data is set correctly. **This registry key must be created manually.**

Key	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Print\Printers
Value Name	DXP01SilentMode
Value Type	REG_DWORD
Data	1 = enable, any other value disable

2. The driver checks the DXP01SilentMode setting at startup.

To guarantee that the setting takes effect, restart the computer after you create or modify the registry setting.

Silent Mode Operation Notes

- Enabling silent mode causes suppression of pop-up messages for all instances (printers) of the XPS Card Printer Driver for all user accounts on the system.
- The SDK application can retrieve the error message any time using dxp01sdk:PRINTER_ MESSAGES. In addition, most of the SDK calls include printer errors as part of the status information returned to the application.
- The application can cancel jobs using the SDK, including canceling all jobs in the printer. When “cancel all jobs” is requested, the printer cancels all of its jobs. The driver also cancels all the driver jobs that are in an error state.

  The printer operator can cancel the job using the LCD panel. When this happens, an error is removed from the driver automatically. Make sure that the application accounts for this possibility.

- When the error is a driver condition (a 500-level message), the application must resolve the error because the printer operator won’t be aware of the issue (the printer does not issue an error). The driver will not process the next job until the 500-level message is resolved. The application can either use “cancel all jobs” to cancel the job, or it can issue job-specific cancel or resume commands to recover from the error.

Appendix E: SDK CE870 Kiosk System Support

E

Overview

The XPS Card Printer Driver supports the CE870 Kiosk system. To use the Card Printer Driver and SDK with the CE870 kiosk system, you need the following:

Firmware	D3.17.5 or newer
XPS Card Printer Driver	7.4.673 or newer
SDK	7.4.097 or newer

The CE870 kiosk system is shipped with the firmware options that support the kiosk already enabled.

For the Card Printer Driver to support the CE870 kiosk system, the driver must be able to recognize the kiosk system. This is controlled by the **DriverEnable** option in the printer firmware. The option is set using one of the following methods:

- The **DriverEnable** option is enabled during manufacturing.
- A secure settings file is used to enable the **DriverEnable** option.

The **DriverEnable** option cannot be changed by a user.

When the **DriverEnable** setting is enabled, the Card Printer Driver uses the Printer Manager WebService-access level option **Printer Setting > EmbossModuleKioskSupport** to detect the kiosk system. The driver configures itself to Kiosk mode and uses the following kiosk DPCL actions:

- Submit action EmbCardDataKiosk instead of EmbCardData
- Send Dispense with no timeout instead of the Eject command

The following Printer Manager settings also are used with a kiosk system. These are WebService-access level settings only. The *CEM Embosser Service Manual* contains complete information about setting up a kiosk system.

Printer Setting > Emboss > KioskRetrieveLocation	Specifies where to place a card that was dispensed via kiosk output, but not taken by the user (from the embosser or from an external card reader), from the following: • **OutputHopper**—Place cards in the embosser output hopper. This is the default setting. • **RejectTray**—Place cards in the embosser reject tray.
Printer Setting > Emboss > KioskWaitDelay	Sets the maximum amount of time that a card is parked inside the kiosk assembly for a user to take until the card is retracted. The default is 60 seconds. When set to 0, the system waits indefinitely for the user to take the card.

When the Card Printer Driver detects a kiosk system:

- The Kiosk option displays in the Printer Properties **Status** tab.
- PRINTER_INFO2 is updated with the Kiosk option. This allows an SDK application to query the printer for the kiosk option.

You can set the Card Printer Driver to run in silent mode. Refer to Appendix D: "Suppress the Driver Message Display”.

Retrieve the Status of a Kiosk Job

An application can retrieve the status of a card job in a kiosk system. The application should poll the kiosk system for job completion until the job state value is returned. The JobState for a kiosk system can have one of the following values: **CardReadyToRetrieve** or **CardNotRetrieved**.

Refer to “Get the Status of an Interactive Job” on page 29 for complete information about how to retrieve the job status and the JobState values for a kiosk system.

Appendix F: Print a UV Photo

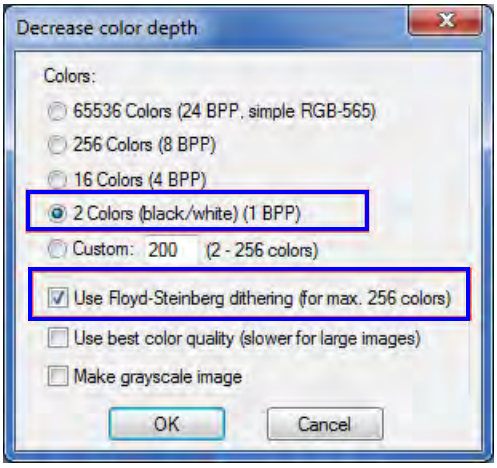
Printing a photo with the UV panel of the ribbon is a special case of monochrome printing. However, unlike printing that uses the K panel, the UV ink is lighter than the card background when illuminated with a UV light source (it becomes the white in the image). Because of this, a photo printed using the UV panel must reverse the pixel values of the 1-bpp photo image (that is, you need to use the negative of the image).

You can use IRFanView (http://www.irfanview.com/) to modify a color photo to print properly using the UV panel.

1. Open the image you want to modify using IRFanView.

2. Select **Decrease Color Depth** from the **Image** menu and then select the following options in the Decrease color depth dialog box.

   - 2 Colors (black/white) (1 BPP)
   - Use Floyd-Steinberg dithering (for max 256 colors)



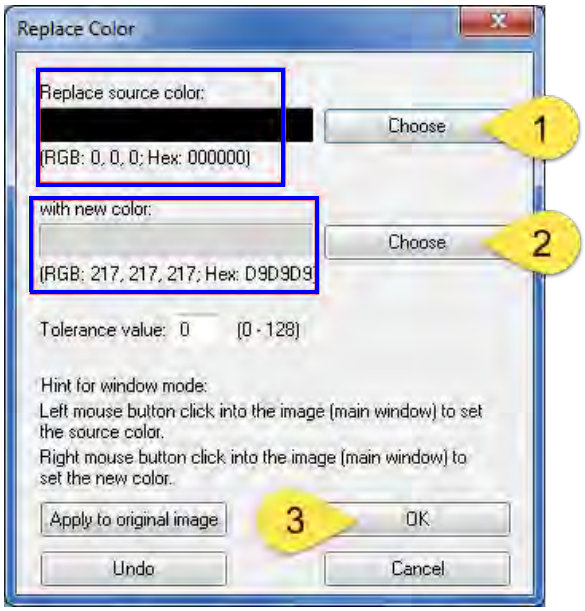
3. Click **OK**.

   This changes the image so that it looks like the following:

4. Select **Negative (invert image)** from the **Image** menu.

   The image then looks like the following. This step is required because the UV ink becomes the "white" of the image when exposed to a UV light source.

5. Select **Replace Color** from the **Image** menu. Replace the source color (black) with the new color (RGB:217,217,217) in the Replace Color dialog box. The Card Printer Driver recognizes 217:217:217 as the UV "color" and prints those elements using the UV panel of the ribbon.



6. Click **OK**.

   The image now looks like the following. The black pixels in the image have been replaced with 217:217:217.

7. Save the converted image as a .PNG file.

- When you use the modified image, be careful not to transform it in any way that might introduce other colors into the image.
- Some images, such as a logo, might display better if they are not reversed.

Appendix G: References

With Microsoft .NET Framework, application developers have a rich set of printing and print system management APIs. At the core of this functionality is the XPS print path. The following link provides an overview of XPS Windows printing:

https://docs.microsoft.com/en-us/dotnet/framework/wpf/advanced/printing-overview

A PrintTicket defines the settings of a print job. A PrintTicket object is an easy-to-work-with representation of a certain type of XML document called a PrintTicket document. The following link explains more about PrintTicket class:

https://msdn.microsoft.com/en-us/library/system.printing.printticket.aspx

You can download Print Schema and XML Paper Specifications using the following link:

https://msdn.microsoft.com/en-us/library/windows/hardware/dn614032(v=vs.85).aspx

Windows has improved bidirectional printer communication (Bidi communication), starting with Windows XP. This allows drivers and applications to make requests to, and get responses from, a printer device. The following link explains more about Bidi printer communication:

https://msdn.microsoft.com/en-us/ff545157

The IBidiSpl interface allows an application to send a Bidi request to the printer. The following link explains more about the IBidiSpl interface:

https://msdn.microsoft.com/en-us/library/dd144980