Datacard®

Datacard® SD, CD, and CE Series Card Printers
Windows Driver Software Development Kit

Programmer's Reference Guide

October 2015

Part No. 527250-001, Rev. G

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

Datacard is a registered trademark and service mark of Entrust Datacard 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 Datacard Corporation. All unauthorized use and reproduction is prohibited.

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

Compliance Statements

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.

Safety

All Datacard® products are built to strict safety specifications in accordance with CSA/UL60950-1 requirements and the Low Voltage Directive 2006/95/EC.

Therefore, safety issues pertaining to operation and repair of **Datacard®** equipment are primarily environmental and human interface.

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.

Regulatory Compliance

Notice for USA (FCC notice)

This equipment has been tested and found to comply with the limits for Class A computing devices, pursuant to Part 15 of FCC rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy. If this equipment is not installed and used in accordance with this instruction manual, it may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at their own expense. Changes or modifications not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment.

Notice for Canada

Industry Canada

This digital apparatus does not exceed the Class A limits for radio noise for digital apparatus set out in the Radio Interference Regulations of the Canadian Department of Communications.

Le présent appareil numérique n'émet pas de bruits radioélectriques dépassant les limites applicables aux appareils numériques de la classe A prescrites dans le Règlement sur le brouillage radioélectrique édicté par le ministère des Communications du Canada.

RSS-Gen, Issue 3, December 2010, Section 7.1.3 User Manual Notice

This Device complies with Industry Canada License-exempt RSS standard(s). Operation is subject to the following two conditions: 1) this device may not cause interference, and 2) this device must accept any interference, including interference that may cause undesired operation of the device.

Cet appareil est conforme avec Industrie Canada RSS standard exemptes de licence(s). Son fonctionnement est soumis aux deux conditions suivantes: 1) ce dispositif ne peut causer des interférences, et 2) cet appareil doit accepter toute interférence, y compris les interférences qui peuvent causer un mauvais fonctionnement du dispositif.

Notice for Europe

The EU Declaration of Conformity can be found on **Datacard.com**

We hereby certify that this printer complies with EMC Directive 2004/108/EC, R&TTE Directive 1999/5/EC, and the EU RoHS Directive EU Directive 2011/65/EC. This printer conforms to Class A of EN 55022 and to EN 301 489-5. Operation of this equipment in a residential environment may possibly cause interference. In the event of interference, the users, at their own expense, will be required to take whatever measures are necessary to correct the problem.

Notice for Europe and Australia

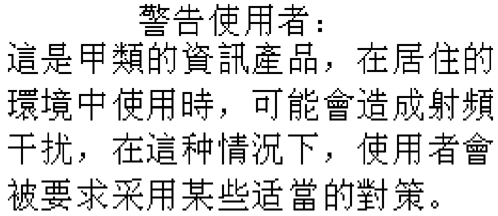
This is a Class A product. In a domestic environment this product may cause radio interference, in which case the user may be required to take adequate measures.

Notice for China (Simplified Chinese)

警告

此为 A 级产品，在生活环境中，
该产品可能会造成无线电干扰。
在这种情况下，可能需要用户
对干扰采取切实可行的措施。

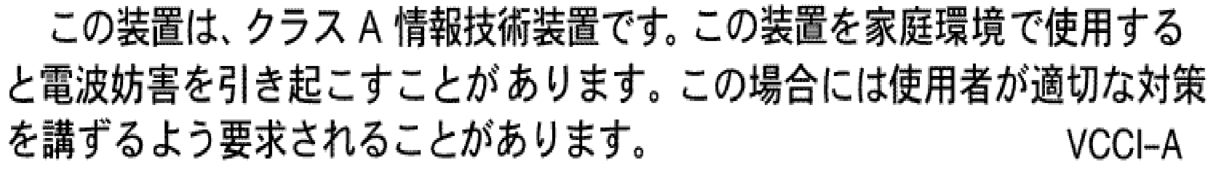
Notice for Taiwan (Traditional Chinese)



警告使用者：
這是甲類的資訊產品，在居住的
環境中使用時，可能會造成射頻
干扰，在這种情況下，使用者會
被要求采用某些适當的對策。

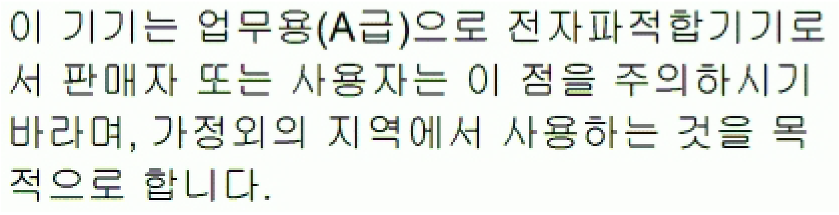
Notice for Japan

Japanese Voluntary Control Council for Interference (VCCI) class A statement



この装置は、クラスA情報技術装置です。この装置を家庭環境で使用する
と電波妨害を引き起こすことがあります。この場合には使用者が適切な対策
を講ずるよう要求されることがあります。 VCCI-A

Korea Communications Commission (KCC) statement



이 기기는 업무용(A급)으로 전자파적합기기로
서 판매자 또는 사용자는 이 점을 주의하시기
바라며, 가정외의 지역에서 사용하는 것을 목
적으로 합니다.

California Proposition 65 Compliance

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.***

Datacard Group believes that its products are not harmful when used as designed. However, the above warning is made in compliance with the State of California Safe Drinking Water and Toxic Enforcement Act of 1986, which requires warning labels on products that may contain elements that the State of California considers harmful.

Revision Log

Windows Driver Software Development Kit Programmer's Reference Guide

Revision	Date	Description of Changes
A	May 2012	First release of this document
B	November 2012	Updates for XPS Card Printer Driver 4.0
C	April 2013	Updates for XPS Card Printer Driver 4.1
D	March 2014	Updates for XPS Card Printer Driver 5.0
E	January 2015	Updates for XPS Card Printer Driver 6.0
F	May 2015	Updates for XPS Card Printer Driver 6.1
G	October 2015	Updates for XPS Card Printer Driver 6.2

Contents

Chapter 1: Introduction

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. Datacard has created a C++ helper 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 E: "References”.

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

- 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 used to select the card
- Use escaped text in the card data to set topcoat blocking rectangles, and set print blocking rectangles
- Use escaped text in the card data to encode standard format magnetic stripe data
- Use escaped text in the card data to specify the input hopper used to select the card
- Use escaped text in the card data to emboss, indent, and top a card when printing to a CE Series printer
- Read magnetic stripe data
- Encode custom magnetic stripe data
- 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
- Laminate, debow, and impress a card
- Process more than one job at a time
- Read the bar code on a serialized overlay
- Check whether the driver or printer is busy and wait before starting a job
- 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
- Restart the printer

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

Installation

For most situations, there are no SDK components to install with your application. You need version 6.2 of the XPS Card Printer Driver and a Datacard SD, CD, or CE Series card printer. A C++ helper DLL is included for Java applications because they 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).

Samples Included in the SDK

The SDK includes eleven samples:

Sample Function	**Code Sample**
Print	print
Magnetic Stripe	magstripe
Smart Card	smartcard
Single-Wire Smart Card	smartcard_singlewire
Single-Wire MIFARE Classic Smart Card	smartcard_singlewire_mifare*
Lamination	lamination*
Laminator Serialized Overlay	lamination_barcode_read*
Emboss and Indent	emboss_indent*
Print Locking	locks*
Printer Control	printer_control*
Status	status

* Not available in Java

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:

- One- or two-sided printing
- Copy count
- Per card-side portrait or landscape orientation (Java is limited to card level orientation)
- Input hopper used to select the card
- Predefined topcoat and print blocking patterns *
- Per card-side 180-degree rotation *
- Per card-side disabling of printing *

* 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
- Input hopper used to select the card using escapes
- Ability to check printer supplies status before printing the card
- Ability to poll for job status and error conditions

Magnetic Stripe Sample

The Magnetic Stripe sample demonstrates magnetic stripe encoding, with options to read the magnetic stripe data, print text on the front of the card, check supplies, and poll for job completion status and error conditions. The print and magnetic stripe data is part of the sample and cannot be changed.

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, check supplies, 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 a 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, check supplies, and poll for job completion status and error conditions. The print data is part of the sample and cannot be changed.

Single-Wire MIFARE 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, check supplies, 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.

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 check supplies and 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 Datacard CE Series system. The emboss and indent data is part of the sample and cannot be changed. The sample also checks supplies, and includes options to specify an input hopper and poll for job completion status and error conditions.

This sample is not available in Java.

Print Locking Sample

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

This sample is not available in Java.

Printer Control Sample

The Printer Control sample demonstrates a way to cancel all jobs in the printer, reset cards counts that are resettable, and restart the printer. 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.

This sample is not available in Java.

Status Sample

The Status sample demonstrates using interactive mode to retrieve printer and supplies information, printer status messages, card counts, job completion status, and error conditions.

Sample Code Location

You can find sample source code under 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 Samples/Outputs folder. These allow you to demonstrate the sample code without your 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, navigate to the appropriate folder and enter the following: sample_name <*printername*> -h.

The compiled samples have the following runtime dependencies.

- **C++:** Requires Microsoft Visual C++ 2013 Redistributable Package (x86 and x64). Use the following link to download the appropriate software package:

  http://www.microsoft.com/en-us/download/details.aspx?id=40784

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

Developer Environments

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

- **C++, C#, and VB.NET:** Microsoft Visual Studio 2013 (You can use any edition, including the free Express Edition, for C# and VB.NET. Visual C++ requires the Professional edition at a minimum.)
- **Java:** Eclipse Helios release. Appendix C: "Using the Java SDK Sample Code with Eclipse" contains step-by-step instructions for importing and building the SDK Java sample code with Eclipse. In addition the Java helper dll requires that the Microsoft Visual C++ 2013 Redistributable Package be installed. The download link is shown in the previous section.

Printing

Your application can either print or block areas on the card from printing and topcoating 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 editor. Java printing does not have access to the Print Ticket, so Java applications are limited to setting the following: orientation (not per-side), one or two-sided, and copy count.

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

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

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. 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 (YMC) 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 (UV) fluorescing (F) panel, text that is 100% opaque and is set at RGB(217,217,217) is rendered by the F 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, a raster graphic is rendered by the monochrome (K) ribbon panel when:
  - 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 (YMC) 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) panel, a raster graphic is rendered by the F 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, 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 (YMC) 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) panel, a vector graphic element is rendered by the F panel when it is 100% opaque and is set to RGB(217,217,217).

Topcoat and Print Blocking

Your card design may have features that must not be printed on or have topcoat applied over. Examples include a contact 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 on using escapes for blocking printing and topcoat can be found in the “Print Blocking Escapes” section of the *Driver Guide*. For more information on non-printing areas, refer to the “Non-Printing Areas” section of the printer’s *Installation and Administrator’s Guide*.

Controlling Card Printing Preferences

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

- 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
- Lamination, debow, and impress actions

Sample Code that Demonstrates Printing

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

Visual C++, Visual C#, and VB.NET	print
Java	javaprint.java
Compiled samples	outputs

View Print Separations

The driver can be configured to redirect the images normally sent to the printer to a file on disk. The output is a zip file which, once extracted, contains a PNG image for color rendering, a PNG image for monochrome rendering, and a PNG image for topcoat. Using these files simplifies the task of confirming that graphics are being separated correctly without using printer supplies. Refer to Appendix B: "Print to a File with the XPS Card Printer Driver” for instructions to configure the driver to print to a file.

The magnetic stripe track data also is written to the zip file, making it a convenient way to inspect the data after it is formatted for the printer.

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 27.

Embossing

Your application can emboss, indent, and apply topping foil to a card with the Datacard CE Card Personalization System by using escapes.

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 your printer's *Driver Guide*.

Embossing Sample Code

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

Visual C++, Visual C#, and VB.NET	emboss_indent
Java	Java does not include an embossing sample at this time.

Laminating

Your application can laminate and impress a card with the Datacard SD460 card printer or a CD800 printer with the optional CLM 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.

Using the Print Ticket to control lamination is the preferred method because it allows your application to control laminating more securely than by using escapes. 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 your printer's *Driver Guide*.

Laminator Bar Code Read

If your CLM laminator 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 37.

Laminating Sample Code

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

Visual C++, Visual C#, and VB.NET	lamination lamination_barcode_read
Java	Java does not include a lamination sample at this time.

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)
- Monitor supplies and printer status
- Single-wire smart card park and personalization
- Read serialized patch overlay bar code
- Monitor and reset card counts
- Get installed printer options
- Lock and unlock a printer with locks
- Restart the printer

Printing, magnetic stripe encoding using escapes or fonts, topcoating, embossing, 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 is provided with the SDK that Java applications can use for interactive mode.

IBidiSpl Requests

The following IBidiSpl requests are used to implement the functions described in the “Overview” on page 19:

Job control (normal)

- Printer.Print:StartJob:Set
- Printer.Print.EndJob:Set
- Printer.Action:Set
- Printer.JobStatus:Read

Job control (error state)

- Printer.PrintMessages:Read
- Printer.Action:Set

Card personalization

- Printer.MagstripeUnit:Back:Encode
- Printer.MagstripeUnit:Back:Read
- Printer.MagstripeUnit:Front:Encode
- Printer.MagstripeUnit:Front:Read
- 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

Printer and supplies capabilities and status

- Printer.PrinterOptions2:Read
- Printer.CounterStatus2:Read
- Printer.SuppliesStatus3:Read
- Printer.ResetCardCount:Set

Laminator

  - Printer.Laminator:BarcodeRead:Set
  - Printer.Laminator:BarcodeReadAndVerify:Set

Printer Control

  - Printer.Restart:Set

Lock control

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

Deprecated—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

Java Helper DLL Interface

The following Java helper DLL functions are used to implement the functions described in the "Overview" on page 19:

Job control (normal)

- StartJob
- EndJob
- ResumeJob
- GetJobStatusXML

Job control (error state)

- CancelJob

Card personalization

- MagstripeEncode2
- MagstripeRead2
- SmartCardPark
- SCardConnect
- SCardDisconnect
- SCardGetAttrib
- SCardStatus
- SCardTransmit

Printer and supplies capabilities and status

- GetPrinterOptions2
- GetPrinterCounterStatus2
- GetPrinterSuppliesStatus

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.

Refer to the sample code for best-practices examples.

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>VISTATEST</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.
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`. For printers with a multi-card input hopper, you can include the input hopper from which to pick the card. For printers other than the SD260, you can check printer and laminator or embosser supplies before starting the job. If nothing is specified, the driver picks a card from hopper 1 and does not check supplies.

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.

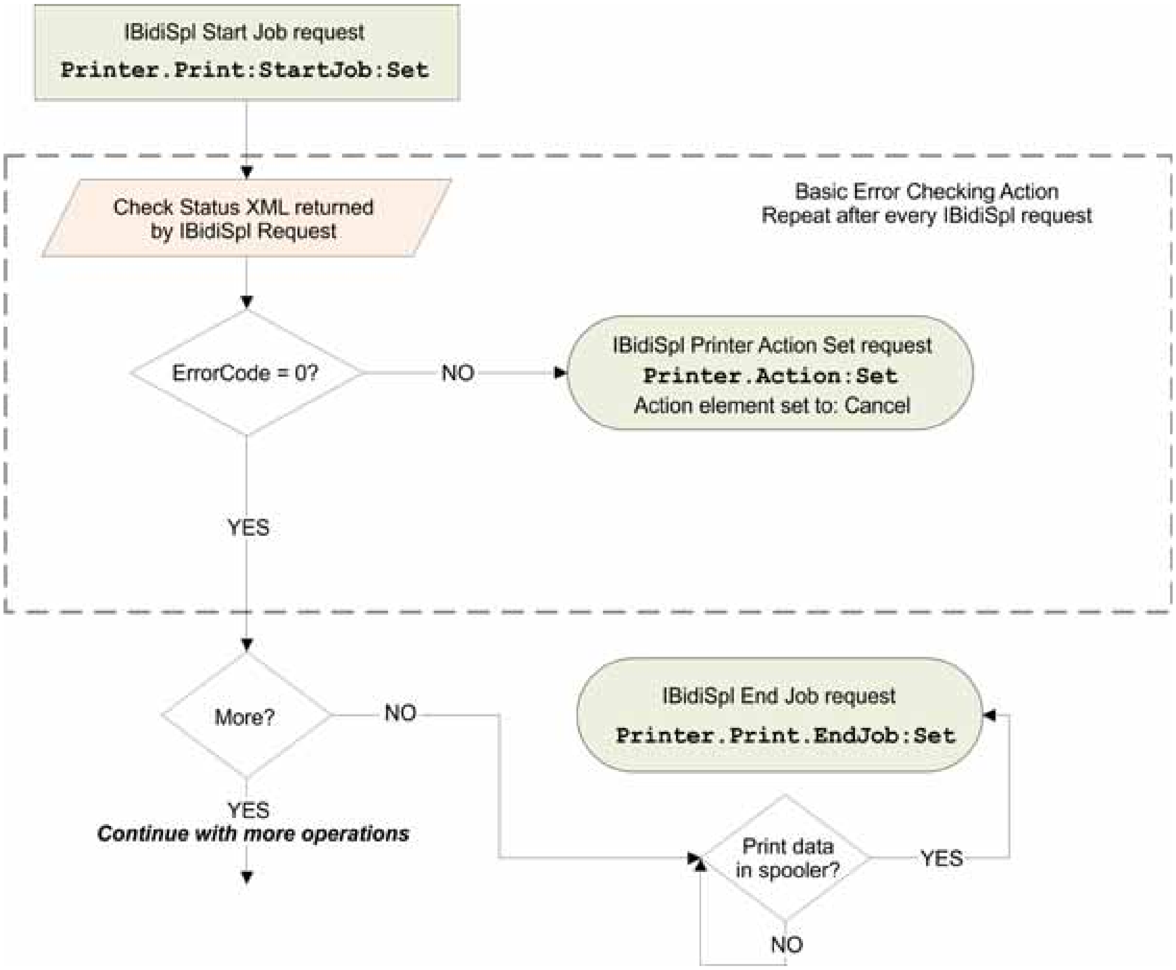
- The input hopper selection and check supplies StartJob options are not supported by Java at this time.
- Using the check supplies option will slow down batch printing.

For Java, call the StartJob 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.


IBidiSpl Start Job request
Printer.Print:StartJob:Set
Check Status XML returned by IBidiSpl Request
Basic Error Checking Action
Repeat after every IBidiSpl request
ErrorCode = 0?
NO
IBidiSpl Printer Action Set request
Printer.Action:Set
Action element set to: Cancel
YES
More?
NO
IBidiSpl End Job request
Printer.Print.EndJob:Set
YES
Continue with more operations
Print data in spooler?
YES
NO

Sample Code

For working code showing 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.java SmartCard.java

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>VISTATEST</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.
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.

JobState value	What it means
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.

Sample Code

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

Visual C++, Visual C#, and VB.NET	magstripe smartcard status
Java	SmartCard.java JobStatusXML.java

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`.

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 will be 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 will be 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, ResumeJob, or SendResponseToPrinter 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 goes to 0, the application can assume that the error was cleared using the printer LCD. Polling the driver for printer messages is not available to Java applications.

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 cancels even those jobs that are not in an error state.

Sample Code

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

Visual C++, Visual C#, and VB.NET	printer_control
Java	Java does not have a sample showing cancel all jobs.

Errors Cleared at the Printer

After an error condition is corrected at the printer, the operator can sometimes 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: "Suppressing 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 back side of 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 printer's *Driver Guide* for details about how to use escapes for magnetic stripe encoding.
- Use the magnetic stripe fonts installed with the XPS 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 printer's *Driver 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

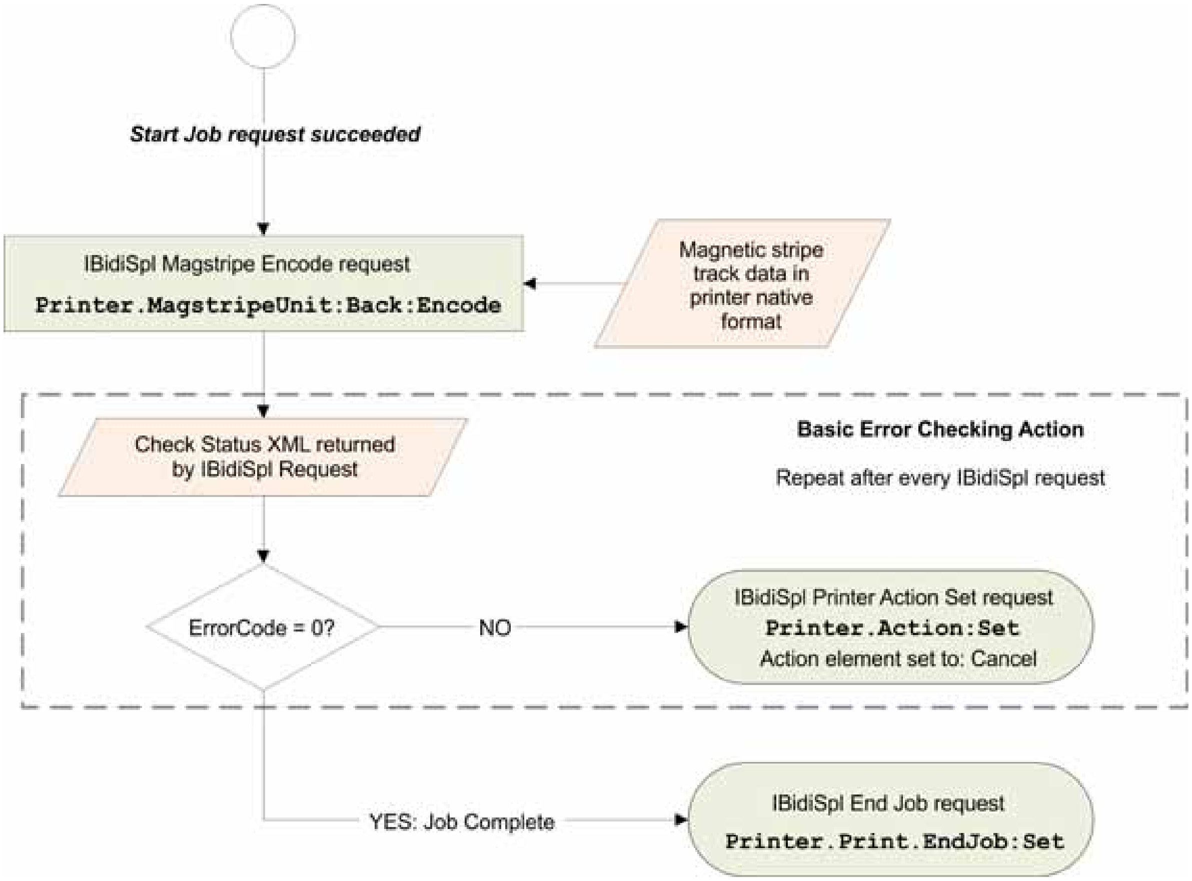
Using the IBidiSpl interface, a card's magnetic stripe can be encoded on the front side or back side of the card. The following assumes you are encoding to the back side of the card.

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:


Start Job request succeeded
IBidiSpl Magstripe Encode request
Printer.MagstripeUnit:Back:Encode
Magnetic stripe track data in printer native format
Check Status XML returned by IBidiSpl Request
Basic Error Checking Action
Repeat after every IBidiSpl request
ErrorCode = 0?
NO
IBidiSpl Printer Action Set request
Printer.Action:Set
Action element set to: Cancel
YES: Job Complete
IBidiSpl End Job request
Printer.Print.EndJob:Set

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 showing interactive mode magnetic stripe encoding, refer to the following samples:

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

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>VISTATEST_{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 (GDI, WinForms, etc.), 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 showing interactive mode magnetic stripe read, refer to the following samples:

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

Read Data from a Serialized Laminate Bar Code

If the CLM laminator 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 either `Printer.Laminator:BarcodeRead:Set` or `Printer.Laminator:BarcodeReadAndVerify:Set`. The BarcodeRead command simply retrieves the bar code data and the card automatically continues. When you use the BarcodeReadAndVerify 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 at this time.

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 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 showing interactive mode smart card station park, refer to the following samples:

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

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\"?>
<!--smartcard connect xml-->
<SmartcardConnect>
    <ProtocolName>SCARD_PROTOCOL_CL</ProtocolName>
</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\"?>
<!--smartcard disconnect xml-->
<SmartcardDisconnect>
       <Method>SCARD_LEAVE_CARD</Method>
</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\"?>
<!--smartcard transmit xml-->
<SmartcardTransmit>
    <Base64Data>O/2RAP+RgXH+QABCAAAAAACBgYAXCACIGQ==</Base64Data>
</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 in the form 0xMMmmbbbb where MM = major version, mm = minor version, and 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 showing personalization of a smart card, refer to the following samples:

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

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

Read and Write Data to MIFARE Classic over Single-Wire

An SDK sample is included that 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.

Sample Code—Single-Wire MIFARE Classic Smart Card Personalization

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

Visual C++, Visual C#, and VB.NET	smartcard_singlewire_mifare
Java	Java does not support this feature at this time.

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

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.

Installed Printer Status, Printer Options, and Supplies Status

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

This request returns the printer status XML file.

```
<?xml version="1.0"?>
<!--Printer options2 xml file.-->
<PrinterInfo2>
<PrinterStatus>Ready</PrinterStatus>
<PrinterAddress>172.16.5.79</PrinterAddress>
<PrinterModel>CD870</PrinterModel>
<PrinterSerialNumber>C15133</PrinterSerialNumber>
<PrinterVersion>D3.12.3-0</PrinterVersion>
<PrinterMessageNumber>0</PrinterMessageNumber>
<ConnectionPortType>Network</ConnectionPortType>
<ConnectionProtocol>Version2Secure</ConnectionProtocol>
<OptionInputhopper>MultiHopper6WithExceptionSlot</OptionInputhopper>
<OptionMagstripe>ISO</OptionMagstripe>
<OptionRewritable>None</OptionRewritable>
<OptionSmartcard>Installed</OptionSmartcard>
<OptionDuplex>Auto</OptionDuplex>
<OptionLock>Installed</OptionLock>
<LockState>Locked</LockState>
<PrintHead>Installed</PrintHead>
<ColorPrintResolution>300x300 | 300x600</ColorPrintResolution>
<MonochromePrintResolution>300x300|300x600|300x1200</MonochromePrintResolution>
<TopcoatPrintResolution>300x300</TopcoatPrintResolution>
<EmbossModule>Installed</EmbossModule>
<EmbosserVersion>E1.1.24-0</EmbosserVersion>
</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.

Message Number

The MessageNumber element contains the printer 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
PrinterAddress	The IP address of the network printer
ConnectionPortType	Identifies the physical connection being used to communicate to the printers. The values are: • Network • USB
ConnectionProtocol	Identifies the protocol used to communicate with the printer. The 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 • SingleHopperWithExceptionSlot • MultiHopper6WithExceptionSlot
OptionMagstripe	The magnetic stripe configuration for this printer. The values are: • None • ISO • JIS
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.

Element Value	**Description**
OptionSmartcard	The smart card configuration for this printer. The values are: • None • Installed • SingleWire
OptionDuplex	The duplex configuration for this printer. The values are: • Manual • Auto
OptionLock	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.
PrintHead	Indicates if this printer includes a print head. (The printer might not have a printhead if you are connected to an emboss-only CE 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 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 This element is missing if the PrintHead value is None.

Element Value	**Description**
TopcoatPrintResolution	The topcoat printing resolutions supported by this printer. At this time, this element always displays the value 300x300. This element is missing if the PrintHead value is None.
EmbossModule	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
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

Sample Code—Printer Status

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

Visual C++, Visual C#, and VB.NET	status
Java	PrinterStatusXML.java

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"?>
<PrinterSupplies3>
    <PrinterStatus>Ready</PrinterStatus>
    <PrintRibbonType>YMCKT</PrintRibbonType>
    <RibbonRemaining>76</RibbonRemaining>
    <RibbonSerialNumber>E0055000008D355F</RibbonSerialNumber>
    <RibbonLotCode>10232012 </RibbonLotCode>
    <RibbonPartNumber>535000003</RibbonPartNumber>
    <IndentRibbon>Installed</IndentRibbon>
    <IndentRibbonRemaining>20</IndentRibbonRemaining>
    <TopperRibbonType>Gold</TopperRibbonType>
    <TopperRibbonRemaining>40</TopperRibbonRemaining>
</PrinterSupplies3>
```

Element Value	Description
PrintRibbonType	The type of ribbon installed in the printer. The value returned will be one of the supported ribbon types for the printer, such as YMCKT, ymcKT, KT, 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.
IndentRibbon	If the system includes an embosser, this element indicates if indent ribbon is installed. The values are: ● None ● Installed
IndentRibbonRemaining	The amount of unused indent ribbon as a percent.

Element Value	**Description**
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.
L1Laminate	Indicates if the laminator L1 station is reporting that it has a supply loaded. The values are: • None • Installed
L1LaminateType	The universal supply code of the supply.
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 is reporting that it has a supply loaded. 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.

Sample Code—Supplies Status

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

Visual C++, Visual C#, and VB.NET	status
Java	PrinterSuppliesStatus.java

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>
<CurrentPicked>15</CurrentPicked>
<TotalPicked>15</TotalPicked>
<CurrentCompleted>14</CurrentCompleted>
<TotalCompleted>14</TotalCompleted>
<CurrentRejected>1</CurrentRejected>
<TotalRejected>1</TotalRejected>
<CurrentLost>0</CurrentLost>
<TotalLost>0</TotalLost>
<CurrentPickedException>0</CurrentPickedException>
<TotalPickedException>0</TotalPickedException>
<CardsPickedSinceCleaningCard>100</CardsPickedSinceCleaningCard>
<CleaningCardsRun>1</CleaningCardsRun>
</CounterStatus2>
```

Status XML for Six-Position Input Hopper Printer

```
<?xml version="1.0"?>
<!--Printer counter2 xml file.-->
<CounterStatus2>
<PrinterStatus>Ready</PrinterStatus>
<CurrentPicked>371</CurrentPicked>
<TotalPicked>371</TotalPicked>
<CurrentCompleted>298</CurrentCompleted>
<TotalCompleted>298</TotalCompleted>
<CurrentRejected>71</CurrentRejected>
<TotalRejected>71</TotalRejected>
<CurrentLost>2</CurrentLost>
<TotalLost>2</TotalLost>
<CurrentPicked1>189</CurrentPicked1>
<TotalPicked1>189</TotalPicked1>
<CurrentPicked2>43</CurrentPicked2>
<TotalPicked2>43</TotalPicked2>
<CurrentPicked3>38</CurrentPicked3>
<TotalPicked3>38</TotalPicked3>
<CurrentPicked4>36</CurrentPicked4>
<TotalPicked4>36</TotalPicked4>
<CurrentPicked5>34</CurrentPicked5>
<TotalPicked5>34</TotalPicked5>
<CurrentPicked6>31</CurrentPicked6>
<TotalPicked6>31</TotalPicked6>
<CurrentPickedException>0</CurrentPickedException>
<TotalPickedException>0</TotalPickedException>
<CardsPickedSinceCleaningCard>100</CardsPickedSinceCleaningCard>
<CleaningCardsRun>1</CleaningCardsRun>
</CounterStatus2>
```

Element Value	Description
CurrentPicked	Number of cards picked by the printer. Can be reset at the printer with proper permission.
TotalPicked	Total number of cards picked by this printer.
CurrentCompleted	Number of cards successfully completed by the printer. 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. 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.

Element Value	**Description**
CurrentLost	A calculated value for the cards that were neither completed nor rejected. Can be reset at the printer with proper permission.
TotalLost	Total number of cards that were neither completed nor rejected.
CurrentPickedException	Number of cards picked from the exception slot. The driver does not provide a means to select the exception slot so this number is typically zero.
TotalPickedException	Total number of cards picked from the exception slot.
CurrentPicked1–CurrentPicked6	Number of cards picked from a specific hopper of a multi-card hopper printer. Can be reset at the printer with proper permission.
TotalPicked1–TotalPicked6	Total number of cards picked from a specific hopper of a multi-card hopper printer.
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 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`.

This function is not available for Java.

Sample Code—Card Counts

For working code showing 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	PrinterCounterStatus.java

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

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 showing the lock operation, refer to the following samples:

Visual C++, Visual C#, and VB.NET	locks
Java	Java does not support locking at this time.

Restart Printer

Your application can restart a printer using the IBidiSpl interface with the schema set to `Printer.Restart:Set`.

Sample Code—Restart Printer

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

Visual C++, Visual C#, and VB.NET	printer_control
Java	Java does not support restart printer at this time.

Interactive Mode Best Practices

- When interactive mode operations are used for card personalization, the driver will not accept a job until the interactive operations for the active job complete. It is the responsibility of your application 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 24.
- Your application should always verify that the printer is online before starting a job. Refer to "Installed Printer Status, Printer Options, and Supplies Status" on page 48 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 will be 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.

Message	Description
118	Print ribbon type problem.
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.
134	Indent ribbon low.
135	Indent ribbon supplies out.
136	Indent ribbon break.
137	Embosser wheel error.
138	Embosser indent error.
139	Card not in position in embosser.
140	Embosser not available.
141	Close emboss cover.

Message	Description
142	Emboss cover error.
143	Topping foil problem.
144	Topping foil out.
145	Topping foil type problem.
146	Topping foil support err.
147	Topping foil no tag found.
148	Topping foil low.
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.
170	Insert new card side 1 up.
171	Insert same card side 1 up.
172	Insert cleaning card.
173	Improper shutdown.
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
500	The printer is not available.

Message	Description
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.
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.

Appendix B: Print to a File with the XPS Card Printer Driver

B

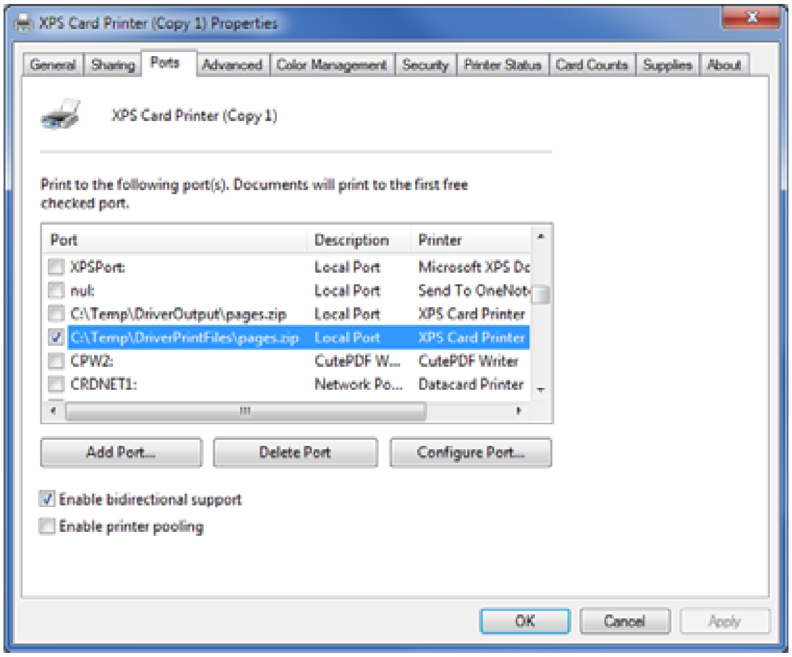
You can “print” without having a printer attached using the XPS Card Printer Driver. The result is a zip file that contains the PNG images that normally would be sent to the printer.

1. Create a folder to hold the files (for example: D:\Temp\DriverPrintFiles). This folder is used in Step 3.
2. Install a network printer (if not already installed). Use the XPS Card Printer installation instructions to install a network printer if one is not already installed. When the Configure Port window displays, follow the instructions in Step 3.
3. Create and assign the printer to a local port. If you are installing a new printer, the installation displays the Ports tab on the Printer Properties window. If a printer is installed already, open the Printer Properties window for the printer and click the Ports tab.
    a. Create a new local port:
        i. Click **Add Port**.
        ii. Select **Local Port** and click **New Port**.
        iii. Enter the path to the folder you created in Step 1 and add the file name: pages.zip. For example: D:\Temp\DriverPrintFiles\pages.zip.

b. Assign the printer to the new local port. The new local port is selected automatically.

  i. Click **Apply**.

  The screen resembles the following.



Print to the following port(s). Documents will print to the first free checked port.

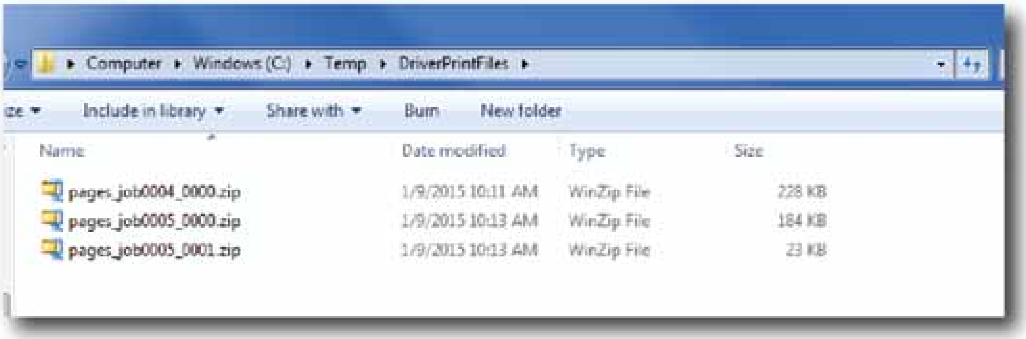
Port	Description	Printer
☐ XPSPort:	Local Port	Microsoft XPS Dc
☐ nul:	Local Port	Send To OneNote
☐ C:\Temp\DriverOutput\pages.zip	Local Port	XPS Card Printer
☑ C:\Temp\DriverPrintFiles\pages.zip	Local Port	XPS Card Printer
☐ CPW2:	CutePDF W...	CutePDF Writer
☐ CRDNET1:	Network Po...	Datacard Printer

☑ Enable bidirectional support

☐ Enable printer pooling

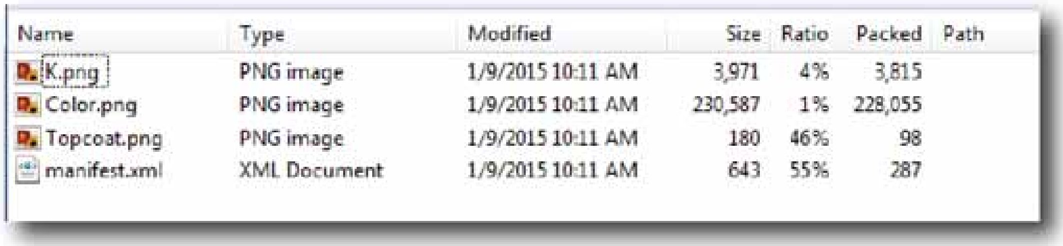
  ii. Click **OK** to close the Printer Properties window.

4. Print as usual. After printing, open the folder you created. A list of files similar to the following displays:



Name	Date modified	Type	Size
pages_job0004_0000.zip	1/9/2015 10:11 AM	WinZip File	228 KB
pages_job0005_0000.zip	1/9/2015 10:13 AM	WinZip File	184 KB
pages_job0005_0001.zip	1/9/2015 10:13 AM	WinZip File	23 KB

5. Inspect the printer-ready image files. Unzip the job you want to inspect. A list of files similar to the following displays:



Name	Type	Modified	Size	Ratio	Packed	Path
K.png	PNG image	1/9/2015 10:11 AM	3,971	4%	3,815	
Color.png	PNG image	1/9/2015 10:11 AM	230,587	1%	228,055	
Topcoat.png	PNG image	1/9/2015 10:11 AM	180	46%	98	
manifest.xml	XML Document	1/9/2015 10:11 AM	643	55%	287	

If you encoded magnetic stripe data, the list will include an additional XML file containing that data formatted for the printer.

Appendix C: Using the Java SDK Sample Code with Eclipse

The XPS Driver SDK Java samples work with either the 32- or 64-bit Java runtimes. Make sure a Java runtime is installed on your computer. From the command line, issue 'java -version':

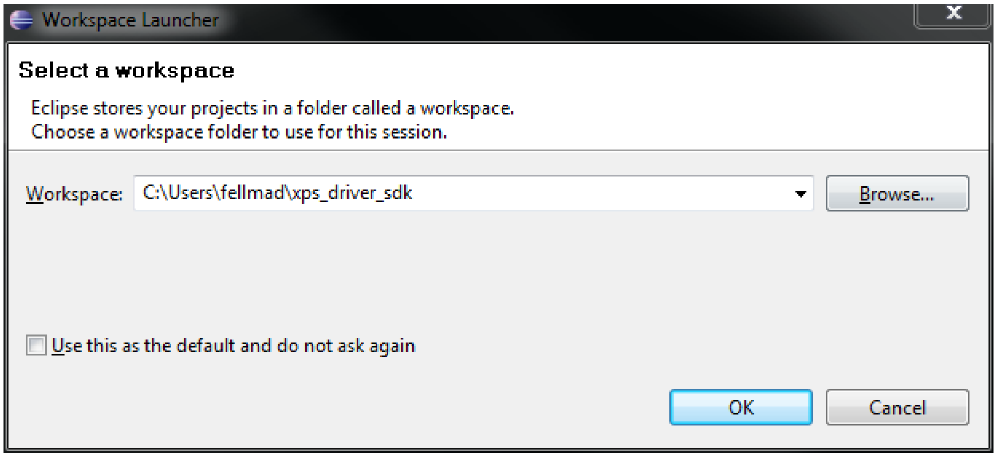
```
C:\Users\fellmad>java -version

    java version "1.6.0_23"
    Java(TM) SE Runtime Environment (build 1.6.0_23-b05)
    Java HotSpot(TM) Client VM (build 19.0-b09, mixed mode, sharing)
```

1. Extract the XPS Driver SDK zip file to a folder. For example: D:\java\xps_driver_sdk:

```
d:\java\xps_driver_sdk>dir
Directory of d:\java\xps_driver_sdk
03/04/2015  03:55 PM    <DIR>          doc
03/02/2015  01:46 PM                21 readme.txt
03/04/2015  03:55 PM    <DIR>          samples
03/04/2015  03:51 PM         1,990,963 XPS_Card_Printer_SDK.zip
```

2. Start Eclipse and create a new workspace.



Select a workspace

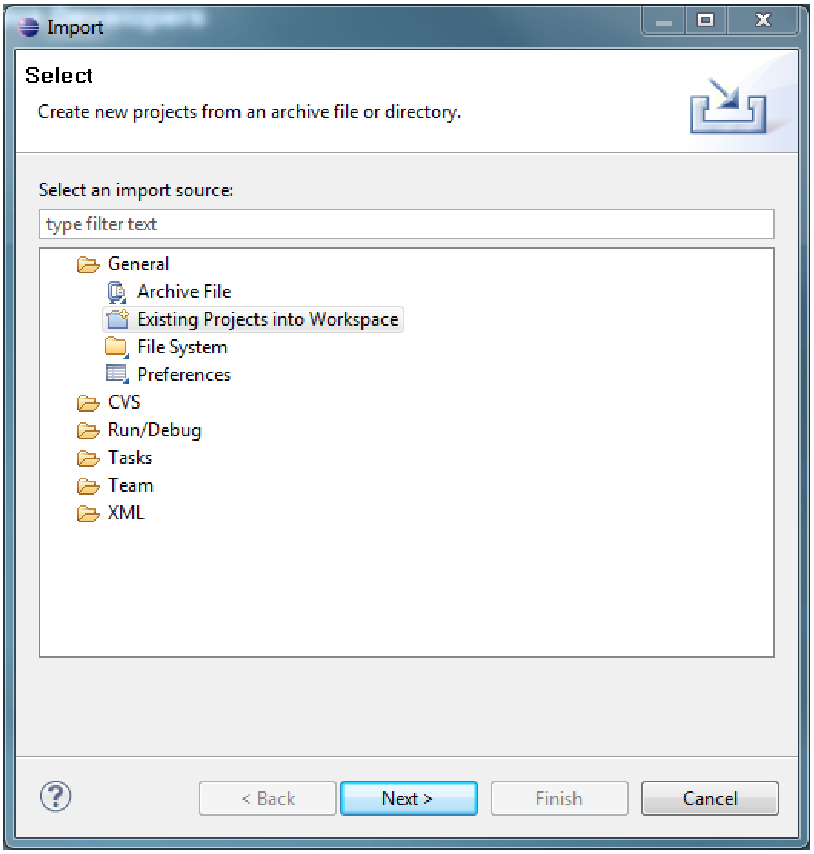
Eclipse stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\Users\fellmad\xps_driver_sdk

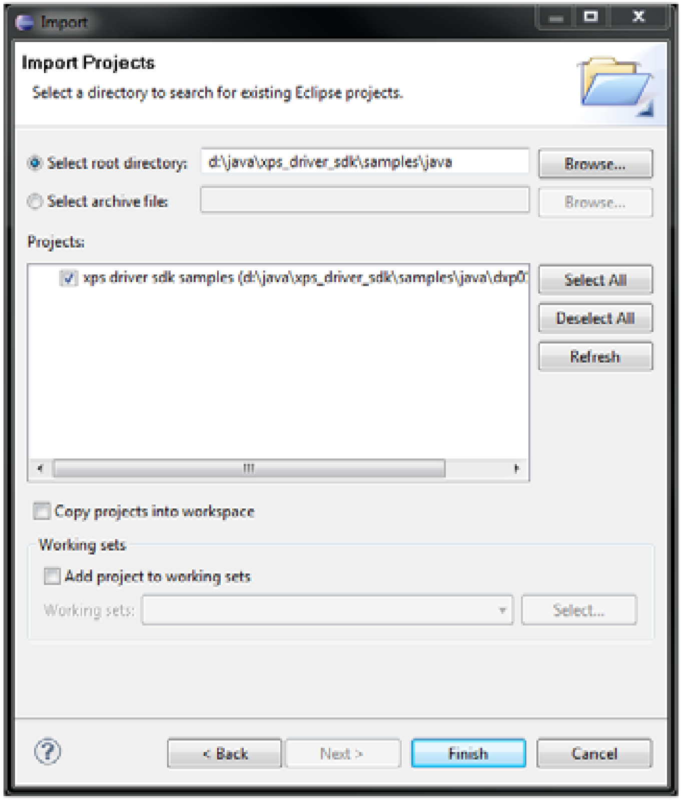
☐ Use this as the default and do not ask again

3. Import the SDK samples.

    a. Select **File | Import and Existing Projects into Workspace**.



b. Click **Next**.



Import Projects

Select a directory to search for existing Eclipse projects.

● Select root directory: d:\java\xps_driver_sdk\samples\java

○ Select archive file:

Projects:

☑ xps driver sdk samples (d:\java\xps_driver_sdk\samples\java\dxp0

☐ Copy projects into workspace

Working sets

☐ Add project to working sets

Working sets:

c. Browse to the 'samples\java' folder under the folder you created in Step 1.

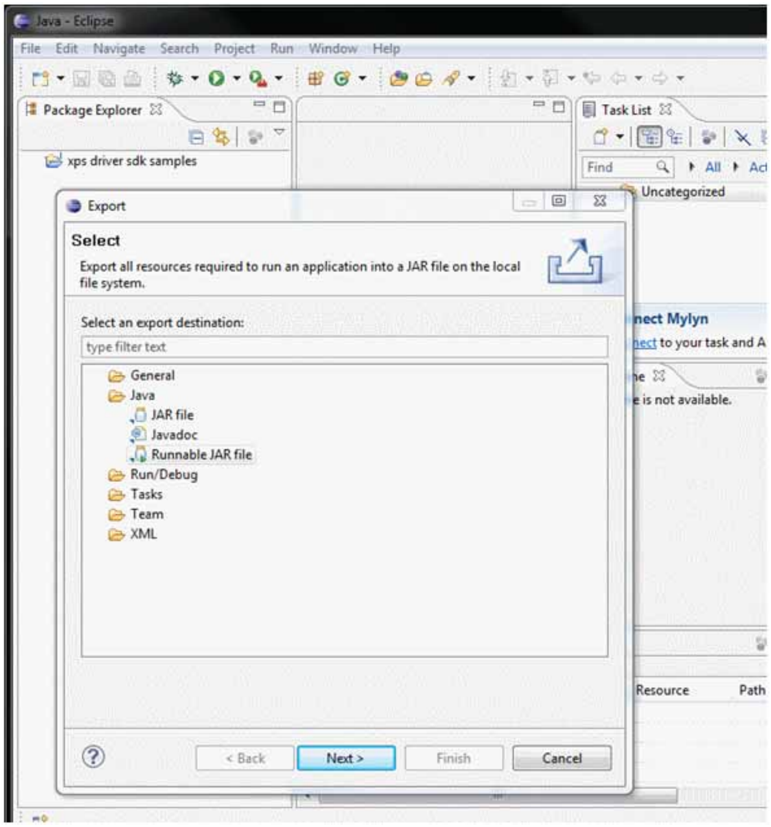
d. Click **Finish**.

4. Create a runnable JAR file.

   a. In the Eclipse Package Explorer, select **com.sun.jna.examples**.

   b. Right-click com.sun.jna.examples and select **Run As**, then **Java Application**.

c. Create a runnable JAR file using **File | Export** and selecting **Java- | Runnable JAR File**:



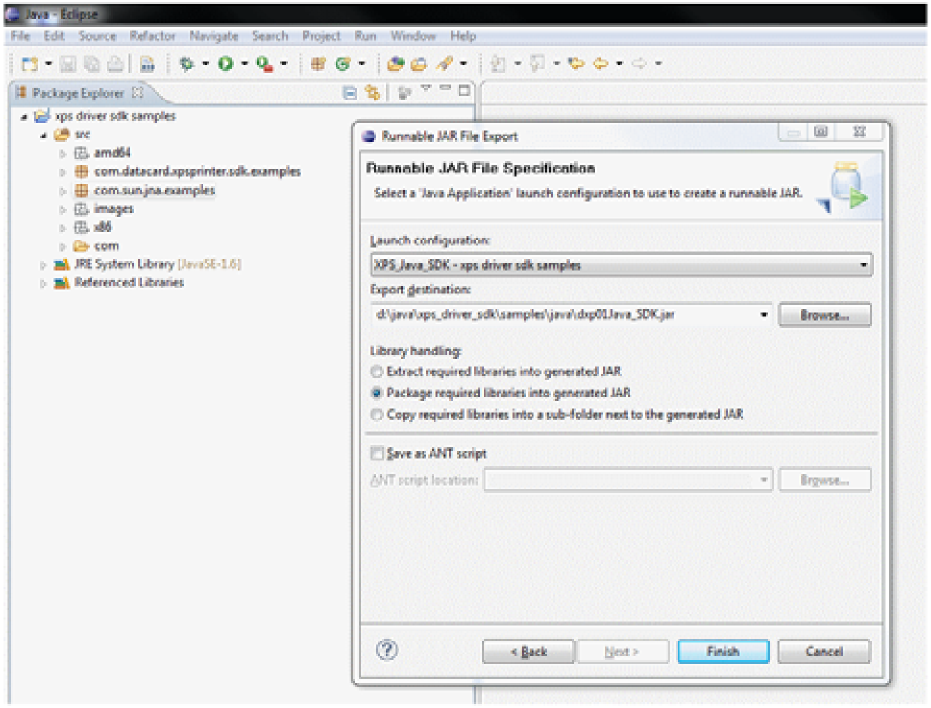
Select

Export all resources required to run an application into a JAR file on the local file system.

Select an export destination:

type filter text

d. When prompted, select **XPS_Java_SDK - xps driver sdk samples** for the Launch configuration. For the JAR filename, use **dxp01Java_SDK.jar**:



Runnable JAR File Specification

Select a 'Java Application' launch configuration to use to create a runnable JAR.

Launch configuration: XPS_Java_SDK - xps driver sdk samples

Export destination: d:\java\xps_driver_sdk\samples\java\dxp01Java_SDK.jar

Library handling:

○ Extract required libraries into generated JAR
● Package required libraries into generated JAR
○ Copy required libraries into a sub-folder next to the generated JAR

☐ Save as ANT script

ANT script location:

This creates a JAR file in the location you specify:

```
d:\java\xps_driver_sdk\samples\java>dir
Directory of d:\java\xps_driver_sdk\samples\java
03/04/2015  04:19 PM    <DIR>          dxp01Java_SDK
03/04/2015  04:28 PM         1,033,154 dxp01Java_SDK.jar
03/04/2015  03:55 PM    <DIR>          Library
```

e. Run the JAR file with no parameters to get help and to see the command line options:

```
d:\java\xps_driver_sdk\samples\java>java -jar dxp01Java_SDK.jar
…
-n <printername>. Required. Try -n "xps card printer"
-p Print                          [l][d][e][number]
-mr Magnetic stripe read          [p][l][d]
-me Magnetic stripe encode        [mr][p][l][d]
-sc Park a smart card             [me][p][l][d][b]
…
```

Appendix D: Suppressing 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.”

Enabling 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 will cancel all of its jobs. The driver will also cancel 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: References

E

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:

http://msdn.microsoft.com/en-us/library/ms742418.aspx

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:

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

http://msdn.microsoft.com/en-us/library/dd183366(v=VS.85).aspx

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

http://msdn.microsoft.com/en-us/library/dd144980(v=VS.85).aspx