ZEBRA SCANNER SDK FOR LINUX
DEVELOPER'S GUIDE

ZEBRA SCANNER SDK FOR LINUX DEVELOPER'S GUIDE

MN-003028-03

Revision A

May 2019

Revision History

Changes to the original guide are listed below:

Change	Date	Description
-01 Rev A	4/2017	Initial release.
-02 Rev A	6/2017	Moved Sample Source Code chapter to Chapter 5 Updated Third Party Software Tools and Libraries Updated Minimum Software Requirements Added Installing CoreScanner and SDK section Updated Table 2-3 with new filenames for CoreScanner version 4.0 Added Verification Methods and Utilities (Install Verifier Application (IVA)) Updated Table 5-1 with new filenames for CoreScanner version 4.0
-03 Rev A	6/2019	Updates: - **OPOS** changed to **UPOS** on pg. 1-1 and in Many Applications Connected to Many Scanner section on pg. 1-6 - Table 1-1 on pg. 1-4 - Note on pg. 2-1 - Table 2-1 on pg. 2-2 - Table 2-2 on pg. 2-3 - new CoreScanner version 4.3 on pgs. 2-3 and 2-7 - Figure 2-1 on pg. 2-4 - section title **Installing CoreScanner and SDK** changed to **Installing and Configuring CoreScanner and SDK** on pg. 2-5 - steps 5 and 6 in Installing the Source Tar Ball on pg. 2-7 - Table 2-3 on pg. 2-8 - CommandResponseEvent and IO sections removed - Table 3-11 (Table 3-10 in previous rev.) on pg. 3-17 - Table 3-14 (Table 3-13 in previous rev.) on pg. 3-36 - **OPOS/IBM OPOS** mode changed to **IBM** mode on pg 4-15 - Quick Startup and Zebra copyright statement of the last page. Added: - Configuration section in Chapter 2 - BinaryDataEvent on pg. 3-14.

TABLE OF CONTENTS

ABOUT THIS GUIDE

Introduction

This guide provides information about the Zebra Scanner Software Developer Kit (SDK) for Linux, an architectural framework providing a single programming interface across multiple programming languages and across multiple system environments for all scanner communication variants (such as SNAPI, IBMHID, HIDKB).

Chapter Descriptions

Topics covered in this guide are as follows:

- *Chapter 1, INTRODUCTION TO THE SCANNER SDK* provides an overview of the Zebra Scanner Software Developer Kit (SDK).
- *Chapter 2, INSTALLATION & CONFIGURATION* describes how to install the Zebra Scanner SDK and components on recommended platforms.
- *Chapter 3, SCANNER SDK API* provides a set of APIs to interface with scanner devices.
- *Chapter 4, TEST UTILITIES & SOURCE CODE* provides information about testing and evaluating Zebra Scanner SDK software components using SDK test utilities.
- *Chapter 5, SAMPLE SOURCE CODE* provides information about how a developer uses the Zebra Scanner SDK.

Notational Conventions

The following conventions are used in this document:

- Courier New font is used for code segments.
- *Italics* are used to highlight:
  - Chapters and sections in this and related documents
  - Dialog box, window and screen names
  - Drop-down list and list box names
  - Screen field names
  - Check box and radio button names
  - File names
  - Directory names.
- **Bold** text is used to highlight:
  - Parameter and option names
  - Icons on a screen
  - Key names on a keypad
  - Button names on a screen.
- bullets (•) indicate:
  - Action items
  - Lists of alternatives
  - Lists of required steps that are not necessarily sequential
- Sequential lists (e.g., those that describe step-by-step procedures) appear as numbered lists.
- Notes, caution and warning statements appear as follows:

NOTE This symbol indicates something of special interest or importance to the reader. Failure to read the note does not result in physical harm to the reader, equipment or data.

CAUTION This symbol indicates that if this information is ignored, the possibility of data or material damage may occur.

WARNING! **This symbol indicates that if this information is ignored the possibility that serious personal injury may occur.**

Service Information

If you have a problem using the equipment, contact your facility's technical or systems support. If there is a problem with the equipment, they will contact the Zebra Technologies Global Customer Support Center at: www.zebra.com/support.

CHAPTER 1 INTRODUCTION TO THE SCANNER SDK

Overview

CoreScanner is the driver which prepares the scanner to be used by the host. The CoreScanner driver creates a connection between the host and scanner. The SDK uses CoreScanner to receive data from and send commands to the scanner.

The Zebra Scanner Software Developer Kit (SDK) for Linux defines an architectural framework providing a single programming interface across multiple programming languages (such as C++, Java) and across multiple system environments (such as RPM based Linux distributions and Debian based Linux) for all scanner communication variants (such as IBMHID, SNAPI, and HIDKB).

The Zebra Scanner SDK includes a suite of components that provides a unified software development framework with a wide range of functions for interfacing Zebra scanners to user applications and solutions.

This SDK allows you to read barcodes, manage scanner configurations, capture images/videos, and select a list of scanners on which to work. While one application in one programming language uses a scanner or a set of scanners, you can use another application in a different language within the same system environment.

For a list of popular topics within this guide, see *Quick Startup*.

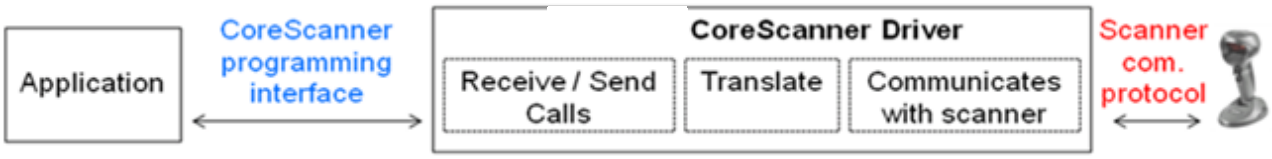


Figure 1-1 *Software Developer Framework*

The SDK can build an application with complete control of scanner capabilities:

- Data, barcode
  - uPOS/JPOS output
  - SNAPI output

- Command and control
  - LED and beeper control
  - Aim control
  - Scan enable and disable
- Imaging
  - Capture and transfer of images
  - Capture and transfer of video

NOTE For a list of a scanner's supported attribute (parameter) numbers and definitions, refer to the scanner Product Reference Guide, available on the Zebra Support website http://www.zebra.com/support. Attributes include configuration parameters, monitored data, and asset tracking information.

FAQs

Q: Can you connect multiple scanners to the CoreScanner Driver?

A: Yes, you can connect multiple scanners simultaneously to one host running the CoreScanner driver.

Q: If two scanners are connected, can you track data by scanner ID?

A: Yes, if scanner X decodes a barcode **123**, it returns to the application a data event containing the data label **123** and the serial number as the scanner ID.

Q: Can you connect multiple applications to the CoreScanner Driver?

A: Yes, you can connect multiple applications simultaneously on one host running the CoreScanner driver. An application can register from an event type (such as barcode, image, video or management) and receives event information plus the originating scanner ID.

Scanner SDK Architecture

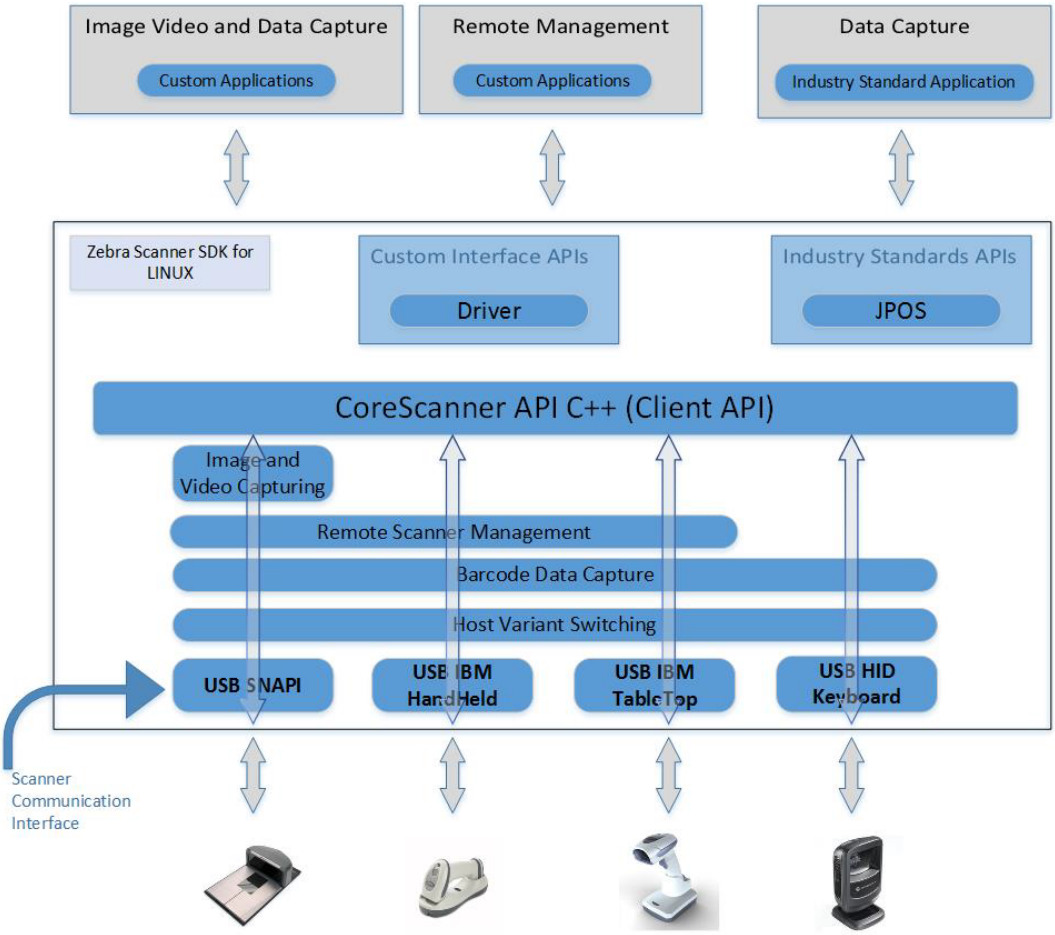


Figure 1-2 *SDK Architecture*

Host Communication Modes

You can configure Zebra scanner devices to operate in different host communication modes such as USB SNAPI, USB OPOS, USB HID Keyboard, and USB IBM Table-top. Device feature support varies by mode but all modes support barcode scanning. Refer to the scanner Product Reference Guide to set the host communication mode.

To use the Zebra Scanner SDK to switch between supported host communication modes, useful when advanced functionality is required but not supported by the current communication mode, call the *Host Variant Switching* command. This switches the scanner to a feature-rich mode to execute certain commands, and then reverts the scanner to the previous mode.

For example, to disable the UPC-A symbology when the device is in USB HID Keyboard mode, switch the scanner to USB SNAPI or USB OPOS (if supported by the scanner), disable UPC-A, and then revert the scanner to USB HID Keyboard mode.

Table 1-1 illustrates scanner capabilities supported in each communication mode. Refer to the scanner specifications for support of each communication mode.

Table 1-1 *Scanner Device Communication Modes Vs. Capabilities*

Capabilities	USB SNAPI	USB HID Keyboard	USB IBM Table-top
Data	Supported	Supported	Supported
Host Variant Switching	Supported	Supported	Supported
Management	Supported	Not Available	Supported
Image & Video	Supported	Not Available	Not Available

Multiple Scanner Device Identification and Asset Tracking

The Zebra Scanner SDK supports connecting multiple scanners to any user application running on CoreScanner APIs. The application identifies each scanner by a unique scanner identification number. There cannot be multiple scanners with the same scanner ID.

Asset tracking information such as model number, serial number, firmware version, and date of manufacture is available if the scanner and its current host mode support the management feature. For example, asset tracking information does not appear for a scanner in USB HID Keyboard mode, but does appear if that scanner is in USB OPOS or USB SNAPI mode.

The format of device asset tracking information can vary between device models. For example, the length of a serial number for DS6707 and DS9808 scanners can be different.

Application Access to Multiple Scanners

The Zebra Scanner SDK supports multiple applications accessing multiple scanners connected to the host at the same time.

The scanner ID uniquely identifies a connected scanner to all applications, and is consistent among all applications for one SDK instance. Restarting the CoreScanner service or the host may assign a different scanner ID to a scanner, but it is unique and referenced by all applications.

Two Applications Connected To One Scanner

App 1 & App 2 support bi-directional (two way) communication with the scanner.

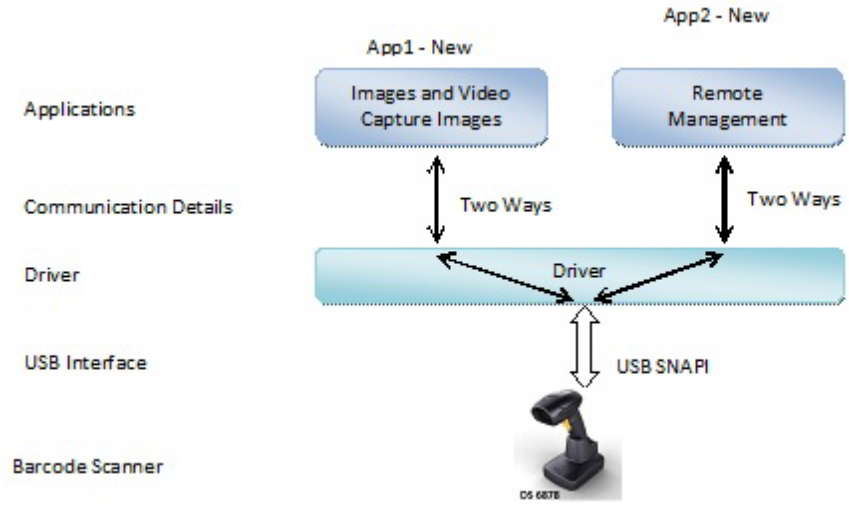


Figure 1-3 *Two Applications Connected To One Scanner*

Three Applications Connected To Two Scanners

App 1 and App 2 support bi-directional (two way) communication with the DS6878.

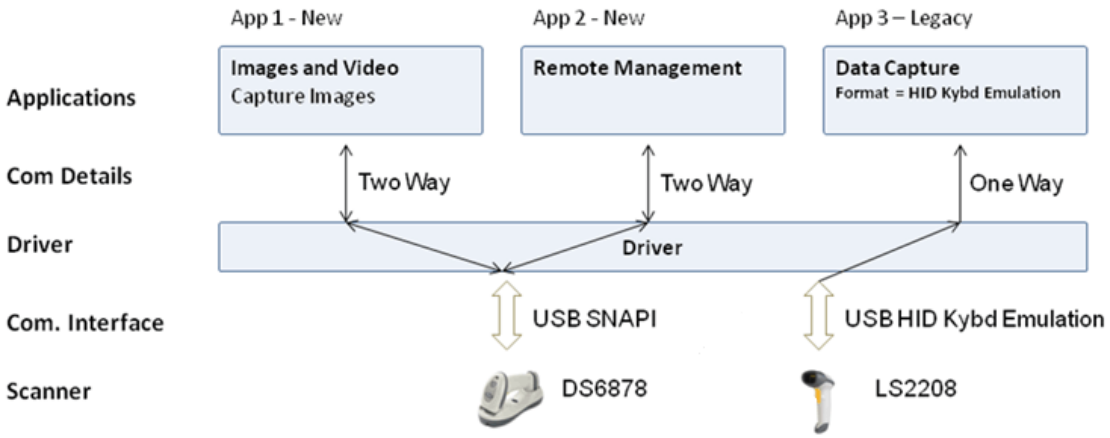


Figure 1-4 *Three Applications Connected To Two Scanners*

Many Applications Connected To Many Scanners

App 1 performs image capture with the DS6878.

App 2 remotely manages both the DS6878 and LS4208.

App 3 receives JPOS data from both the DS6878 and LS4208.

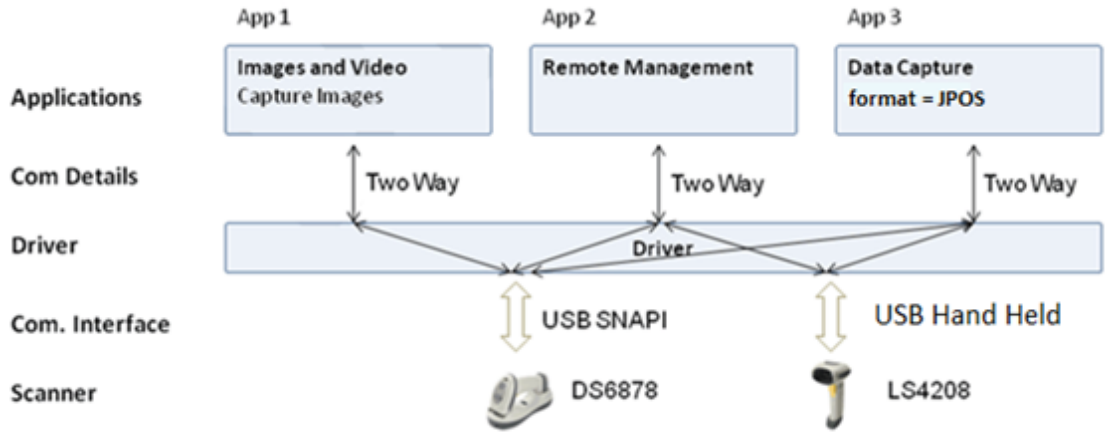


Figure 1-5 *Many Applications Connected To Many Scanners*

One Application Connected to Two Scanners

One application can manage multiple scanners in multiple communication interfaces. The application can capture data, image, and video, send management commands, and receive responses from multiple scanners.

All scanner responses consist of details (ScannerID, serial number, model number, etc.) identifying the scanner that sent the response.

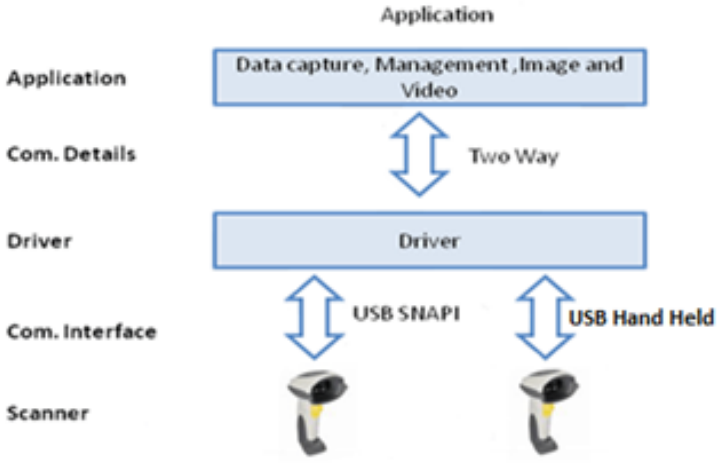


Figure 1-6 *One Application Connected to Two Scanners*

The following example shows a barcode event for a scanned label. The label data contains a unique ScannerID and the scanner's model number and serial number.

```
<?xml version="1.0" encoding="UTF-8"?>
<outArgs>
  <scannerID>1</scannerID>
  <arg-xml>
    <scandata>
      <modelnumber>DS6707-SR20001ZZR </modelnumber>
      <serialnumber>7114000503308   </serialnumber>
      <GUID>9008A01BB72BA34BB519806B63701AAA</GUID>
      <datatype>11 </datatype>
      <datalabel>0x39 0x37 0x38 0x30 0x32 0x30 0x31 0x34</datalabel>
      <rawdata>0x39 0x37 0x38 0x30 0x32 0x30 0x31 0x34</rawdata>
    </scandata>
  </arg-xml>
</outArgs>
```

CHAPTER 2 INSTALLATION & CONFIGURATION

Overview

This chapter describes how to install the Zebra Scanner SDK and components on recommended platforms.

NOTE See *System Requirements on page 2-2* for supported platforms.

The SDK installation package includes:

- Components that enable any Zebra scanner to communicate with applications or tools that execute on top of the Zebra Scanner SDK.
- Supporting documents.
- Test utilities.
- Sample applications and source projects.

NOTE Uninstall any previous Zebra, Symbol, or drivers or SDKs which communicate with Zebra scanners before installing the Zebra Scanner SDK. This includes but is not limited to Zebra and Symbol supplied JPOS driver.

For a list of popular topics within this guide, see *Quick Startup*.

NOTE For a list of a scanner's supported attribute (parameter) numbers and definitions, refer to the scanner Product Reference Guide, available on the Zebra Support website http://www.zebra.com/support. Attributes include configuration parameters, monitored data, and asset tracking information.

SDK Components

The SDK installation package installs the following SDK components to the default locations in *Table 2-3 on page 2-8*:

- Zebra Scanner SDK core components and drivers (USB API (libudev and libusb), imaging drivers)
- JPOS drivers (scanner and scale)
- Link to the latest developer's guide
- Test and sample utilities with source code packages:
  - Scanner SDK GTK Sample Application (C++)
  - Scanner SDK Console Application (C++)
  - JPOS Test Utility for Scanner and Scale

System Requirements

Supported Operating Systems

Table 2-1 *Supported Operating Systems*

Distribution	Version and Architecture
Ubuntu	12.04 (64bit and 32bit) 14.04 (64bit and 32bit) 16.04 (64bit and 32bit)
Debian	7 (64bit and 32bit) 8 (64bit and 32bit)
Suse Linux Enterprise Server	11 SP1 (64bit and 32bit) 11 SP2 (64bit and 32bit) 11 SP3 (64bit and 32bit) 12 (64bit) 12 SP2 (64bit)
RedHat	6.3 (64bit and 32bit) 7.4 (64bit)
CentOS	6.3 (64bit and 32bit) 7 (64bit)
OpenSuse	13.1 (32bit)

Hardware Requirements

Minimum hardware requirements for the Zebra scanner driver and SDK depend on the minimum system requirement for the Linux distribution installation. The following requirements are provided as a guideline.

- Processor: Pentium* 4 1.6 GHz or higher (Pentium 4 2.4 GHz or higher or any AMD64 or Intel64 processor recommended)
- Main memory: 1 GB physical RAM (2 GB recommended)
- Hard disk: 10 GB available disk space for a minimal install, 25 GB available for a graphical desktop (more recommended)

Third Party Software Libraries

Table 2-2 *Minimum Software Requirements for CoreScanner Installation*

Library	Version Limitation
Libudev	147 or above
GTK (for GUI applications)	2.24.10
Java	6 (JDK 1.6)

Scanner Communication Modes

Refer to the scanner's Product Reference Guide for supported communication modes, and specifically the appendix *Functionality Supported via Communication (Cable) Interface* for supported scanner functionality by communication protocol.

CoreScanner and SDK Installers

The CoreScanner Version 4.3 SDK and later offer three types of installers to accommodate many different Linux distributions:

- RPM package-based installers - Installable only on RPM-based Linux distributions
- Debian package-based installers - Installable only on Debian-based Linux distributions
- Source tarball based installers - Installable on any Linux distribution. You are prompted to install all required dependent packages when configuring the source tarball extraction for installation.

Figure 2-1 lists supported distributions for the various installation options.

Installation Options	Supported Distribution															
	CentOS 7	CentOS 6.3	SLES 11 SP3	SLES 11 SP2	SLES 12	Open Suse 13.1	Red Hat 6.3	Red Hat 7.4	Ubuntu 16.04	Ubuntu 14.04	Ubuntu 12.04	Debian 7	Debian 8	Rasbian 6.0	Other *	
RPM_Package_x86_64bit	X	X	X	X	X	X	X	X	NA	NA	NA	NA	NA	NA	NA	X86 Architecture
RPM_Package_x86_32bit	NA	X	X	X	NA	NA	X	NA	NA	NA	NA	NA	NA	NA	NA	X86 Architecture
Debian_Package_x83_64bit	NA	NA	NA	NA	NA	NA	NA	NA	X	X	X	X	X	NA	NA	X86 Architecture
Debian_Package_x83_32bit	NA	NA	NA	NA	NA	NA	NA	NA	X	X	X	X	X	NA	NA	X86 Architecture
Debian	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	coming	NA	ARM Architecture
Source_Tar_GZ_Package	X	X	NA	NA	NA	X	NA	NA	NA	X	X	X	X	NA	X	X86 Architecture

* Customer Testing Required

Figure 2-1 *Supported Distributions*

Choosing an Installer

There are many different Linux bases/trees available. Perform the appropriate research or contact Zebra support to determine the base tree of your distribution. Select the CoreScanner and SDK installer based on the following categories:

- RPM-based packages (.rpm) for the following distributions:
  - CentOS
  - Suse (Open Suse, Suse Linux Enterprise Server, etc.)
  - Opensus
  - Red Hat
  - Fedora
- Debian-based packages (.dpkg) for the following distributions:
  - Debian
  - Ubuntu
- Source tar ball based installer for other Linux distributions such as Gentoo.

NOTE RPM and Debian are package managers supported for the Linux distributions above. Each package manager provides a shell command (rpm or dpkg) for installing/removing packages in a Linux distribution.

RPM and Debian Installer Packages

RPM and Debian installers provide three packages for installing different components:

- **zebra-scanner-corescanner** - The core package of CoreScanner which includes the CoreScanner daemon and libraries. This package places files into the standard file installation locations of Linux systems (in this case /usr/share/zebra-scanner/etc.) Install this package before installing the JavaPOS and Corescanner API development packages, as it is required for both.
- **zebra-scanner-javapos** - Part of the CoreScanner-SDK, this includes all JPOS libraries, sample applications, scripts, and C++ libraries for JNI. The zebra-scanner-corescanner package is required for JPOS package development and implementation.

- **zebra-scanner-devel** - This provides the application programming interface exposed from CoreScanner, and contains header files required to develop C/C++ applications for CoreScanner and source files for simple console based sample applications. The zebra-scanner-corescanner package is required for this API development package.

Installing and Configuring CoreScanner and SDK

This section describes how to install each CoreScanner and SDK package after selecting the suitable installer for the Linux distribution.

Installing RPM Packages

1. Check for existing CoreScanner and SDK installations using the rpm command -qa option:

   rpm -qa | grep zebra

   For example, list packages installed in a CentOS/Suse RPM based distribution:

   [root@localhost]# rpm -qa | grep zebra

   zebra-scanner-devel-3.0.0-0

   zebra-scanner-javapos-3.0.0-0

   zebra-scanner-corescanner-3.0.0-0

2. Remove existing CoreScanner and SDK packages using the rpm command -e option. Since the zebra-scanner-corescanner package is required for JPOS and API development packages, remove this after uninstalling the other two packages.

   For example:

   # rpm -e zebra-scanner-javapos-2.0.0-0

   # rpm -e zebra-scanner-devel-2.0.0-0

   # rpm -e zebra-scanner-corescanner-2.0.0-0

3. Install new packages using the rpm command -i option. Install the CoreScanner packages first to avoid possible package dependency issues.

   For example:

   # rpm -i zebra-scanner-corescanner-2.0.0-0-sles11_sp2-i586.rpm

   # rpm -i zebra-scanner-devel-2.0.0-0-sles11_sp2-i586.rpm

   # rpm -i zebra-scanner-javapos-2.0.0-0-sles11_sp2-i586.rpm

Installing Debian Packages

1. Check for existing CoreScanner and SDK installations using the dpkg command -l option:

   dpkg -l | grep zebra

   For example, list packages installed in a Ubuntu distribution:

   ```
   ii  zebra-scanner-corescanner   2.0.0-0  amd64  Zebra Technologies Linux Corescanner Daemon
   ii  zebra-scanner-devel         2.0.0-0  amd64  Development files for Zebra Technologies Linux
                                                  Corescanner
   ii  zebra-scanner-javapos       2.0.0-0  amd64  Jpos 1.13 drivers for Zebra Technologies Linux
                                                  Corescanner Daemon
   ```

2. Remove existing CoreScanner and SDK packages using the dpkg command -r option. Since the zebra-scanner-corescanner package is required for JavaPOS and API development packages, remove this after uninstalling the other two packages.

   For example:

   ```
   # dpkg -r zebra-scanner-javapos
   (Reading database ... 219367 files and directories currently installed.)
   Removing zebra-scanner-javapos (2.0.0-0) ...
   # dpkg -r zebra-scanner-devel
   (Reading database ... 219343 files and directories currently installed.)
   Removing zebra-scanner-devel (2.0.0-0) ...
   # dpkg -r zebra-scanner-corescanner
   (Reading database ... 219328 files and directories currently installed.)
   Removing zebra-scanner-corescanner (2.0.0-0) ...
   update-rc.d: /etc/init.d/cscored exists during rc.d purge (use -f to force)
   Processing triggers for libc-bin (2.19-0ubuntu6.4) ...
   ```

3. Install new packages using the dpkg -i option. Install the CoreScanner package first to avoid possible package dependancy issues.

   For example:

   ```
   # dpkg -i zebra-scanner-corescanner_2.0.0-0_ubuntu-14.04_amd64.deb
   ```

   Sample output:

   ```
   Selecting previously unselected package zebra-scanner-corescanner.
   (Reading database ... 219288 files and directories currently installed.)
   Preparing to unpack zebra-scanner-corescanner_2.0.0-0_ubuntu-14.04_amd64.deb ...
   Unpacking zebra-scanner-corescanner (2.0.0-0) ...
   Setting up zebra-scanner-corescanner (2.0.0-0) ...
   Adding system startup for /etc/init.d/cscored ...
   /etc/rc0.d/K20cscored -> ../init.d/cscored
   /etc/rc1.d/K20cscored -> ../init.d/cscored
   /etc/rc6.d/K20cscored -> ../init.d/cscored
   /etc/rc2.d/S20cscored -> ../init.d/cscored
   /etc/rc3.d/S20cscored -> ../init.d/cscored
   /etc/rc4.d/S20cscored -> ../init.d/cscored
   /etc/rc5.d/S20cscored -> ../init.d/cscored
   Processing triggers for ureadahead (0.100.0-16) ...
   ```

```
Processing triggers for libc-bin (2.19-0ubuntu6.4) ...
# dpkg -i zebra-scanner-devel_2.0.0-0_ubuntu-14.04_amd64.deb
```

Sample output:

```
Selecting previously unselected package zebra-scanner-devel.
(Reading database ... 219320 files and directories currently installed.)
Preparing to unpack zebra-scanner-devel_2.0.0-0_ubuntu-14.04_amd64.deb ...
Unpacking zebra-scanner-devel (2.0.0-0) ...
Setting up zebra-scanner-devel (2.0.0-0) ...
# dpkg -i zebra-scanner-javapos_2.0.0-0_ubuntu-14.04_amd64.deb
```

Sample output:

```
Selecting previously unselected package zebra-scanner-javapos.
(Reading database ... 219335 files and directories currently installed.)
Preparing to unpack zebra-scanner-javapos_2.0.0-0_ubuntu-14.04_amd64.deb ...
Unpacking zebra-scanner-javapos (2.0.0-0) ...
Setting up zebra-scanner-javapos (2.0.0-0) ...
Processing triggers for libc-bin (2.19-0ubuntu6.4) ....
```

Installing the Source Tar ball

1. Download zebra-scanner-4.0.0.tar.gz to the home directory. This source tarball contains a script file to install CoreScanner and SDK on the Linux system.
2. Extract the zebra-scanner-4.0.0.tar.gz file, which creates the zebra-scanner-4.0.0 directory.

   $> tar -xvf zebra-scanner-4.0.0.tar.gz
3. Change directory to the zebra-scanner-4.0.0 folder.
4. Change the user to a super user using sudo/su commands. A super user password is required.
5. Configure CoreScanner for the Linux distribution, specifying the destination directory as the root directory using the destdir option.

   $> configure --destdir =/
6. Use the “make” build command to build the CoreScanner and SDK for the system. “make” searches for all dependent third-party libraries, and displays errors and aborts the build if it can’t find a dependent package in the environment. Restart installation by installing these packages.

   $> make
7. Use "make install" to install CoreScanner and SDK components.

   $> make install
8. Use "make uninstall" to remove an existing source tarball installation from the system.

   $> make uninstall

Verifying CoreScanner and SDK Installation

CoreScanner version 4.3 and later include the Install Verify Application, which verifies the status of the CoreScanner installation and execution. See *Install Verifier Application (IVA) Utility on page 4-1*.

Installed Components

CoreScanner daemon service (cscore) is installed with the zebra-scanner-corescanner package. This coordinates activity between the communication layer (SNAPI, IBMHH, IBMTT, etc.) and upper level drivers (JPOS, SDK API, etc.).

Table 2-3 lists the components installed.

Table 2-3 *Zebra Scanner SDK Components*

Component	File	Description	Installation Path
CoreScanner	cscore	Scanner Driver Daemon Service	/usr/bin/
CoreScanner	libcs-iudev.so	Transport Component	/usr/lib/zebra-scanner/corescanner
CoreScanner	libcs-iudev.so.4.0.0	Transport Component	/usr/lib/zebra-scanner/corescanner
CoreScanner	libcscl-hidkb.so	Transport Component	/usr/lib/zebra-scanner/corescanner
CoreScanner	libcscl-hidkb.so.4.0.0	Transport Component	/usr/lib/zebra-scanner/corescanner
CoreScanner	libcscl-ibmhh.so	Transport Component	/usr/lib/zebra-scanner/corescanner
CoreScanner	libcscl-ibmhh.so.4.0.0	Transport Component	/usr/lib/zebra-scanner/corescanner
CoreScanner	libcscl-ibmtt.so	Transport Component	/usr/lib/zebra-scanner/corescanner
CoreScanner	libcscl-ibmtt.so.4.0.0	Transport Component	/usr/lib/zebra-scanner/corescanner
CoreScanner	libcscl-snapi.so	Transport Component	/usr/lib/zebra-scanner/corescanner
CoreScanner	libcscl-snapi.so.4.0.0	Transport Component	/usr/lib/zebra-scanner/corescanner
CoreScanner	libcs-client.so	Transport Component	/usr/lib/zebra-scanner/corescanner
CoreScanner	libcs-client.so.4.0.0	Transport Component	/usr/lib/zebra-scanner/corescanner
CoreScanner	libcs-common.so	Transport Component	/usr/lib/zebra-scanner/corescanner
CoreScanner	libcs-common.so.4.0.0	Transport Component	/usr/lib/zebra-scanner/corescanner
CoreScanner	libcs-comm.so	Transport Component	/usr/lib/zebra-scanner/corescanner
CoreScanner	libcs-comm.so.4.0.0	Transport Component	/usr/lib/zebra-scanner/corescanner
CoreScanner	libcs-clientscanner.so	Transport Component	/usr/lib/zebra-scanner/corescanner
CoreScanner	libcs-clientscanner.so. 4.0.0	Transport Component	/usr/lib/zebra-scanner/corescanner
CoreScanner	libcs-clientscale.so	Transport Component	/usr/lib/zebra-scanner/corescanner
CoreScanner	libcs-clientscale.so.4.0.0	Transport Component	/usr/lib/zebra-scanner/corescanner
CoreScanner-JNI	libcs-jniscanner.so.4.0.0	JPOS-JNI Component	/usr/lib/zebra-scanner/javapos/jni
CoreScanner-JNI	libcs-jniscanner.so	JPOS-JNI Component	/usr/lib/zebra-scanner/javapos/jni

Table 2-3 *Zebra Scanner SDK Components (Continued)*

Component	File	Description	Installation Path
CoreScanner-JNI	libcs-jniscale.so.4.0.0	JPOS-JNI Component	/usr/lib/zebra-scanner/javapos/jni
CoreScanner-JNI	libcs-jniscale.so	JPOS-JNI Component	/usr/lib/zebra-scanner/javapos/jni
JPOS	jpostest.sh	Start Script for JPOS Test Application	/usr/lib/zebra-scanner/javapos/jpos
JPOS	dio.sh	Start Script for JPOS Direct IO Test Application	/usr/lib/zebra-scanner/javapos/jpos
JPOS	xml-apis.jar	Third party Java library used for JPOS	/usr/lib/zebra-scanner/javapos/jpos
JPOS	xercesImpl.jar	Third party Java library used for JPOS	/usr/lib/zebra-scanner/javapos/jpos
JPOS	JposTest.jar	JPOS sample application	/usr/share/zebra-scanner/samples/jpos-sample-app
JPOS	JposTestDio.jar	JPOS direct I/O sample application	/usr/share/zebra-scanner/samples/jpos-directio-app
JPOS	JposServiceScanner.jar	JPOS Library	/usr/lib/zebra-scanner/javapos
JPOS	JposServiceScale.jar	JPOS Library	/usr/lib/zebra-scanner/javapos
JPOS	JposServiceOnScanner.jar	JPOS Library	/usr/lib/zebra-scanner/javapos
JPOS	JposServiceOnScale.jar	JPOS Library	/usr/lib/zebra-scanner/javapos
JPOS	JposServiceJniScanner.jar	JPOS Library	/usr/lib/zebra-scanner/javapos
JPOS	JposServiceJniScale.jar	JPOS Library	/usr/lib/zebra-scanner/javapos
JPOS	jpos113.jar	JPOS Library	/usr/lib/zebra-scanner/javapos
Console Application	Makefile	Local Make File for Console Application	/usr/share/zebra-scanner/samples/console-app
Console Application	ConsoleApplication	Execute File of Console Application	/usr/share/zebra-scanner/samples/console-app
Console Application	ConsoleSampleEvent Listener.h	Console Application Source File	/usr/share/zebra-scanner/samples/console-app
Console Application	ConsoleMain.h	Console Application Source File	/usr/share/zebra-scanner/samples/console-app
Console Application	ConsoleSampleEvent Listener.cpp	Console Application Source File	/usr/share/zebra-scanner/samples/console-app
Console Application	ConsoleMain.cpp	Console Application Source File	/usr/share/zebra-scanner/samples/console-app

Table 2-3 *Zebra Scanner SDK Components (Continued)*

Component	File	Description	Installation Path
Configuration	corescanner-sdk.conf	Configurations for CoreScanner and client libraries	/etc/zebra-scanner
Init Script	cscored	Systemv Init Script	/etc/init.d/
CoreScanner	CsBarcodeTypes.h	Header files required for application development on top of CoreScanner	/usr/include/zebra-scanner
CoreScanner	CsUserDefs.h	Header files required for application development on top of CoreScanner	/usr/include/zebra-scanner
CoreScanner	CsIEventListenerXml.h	Header files required for application development on top of CoreScanner	/usr/include/zebra-scanner
CoreScanner	Cslibcorescanner_xml.h	Header files required for application development on top of CoreScanner	/usr/include/zebra-scanner

Configuration

To configure logging, udev options and client library options for CoreScanner and SDK, use the corescanner-sdk.conf file located in the directory /etc/zebra-scanner. See *Configure Logging*, *Configure Udev Options* and *Configure Client Library Options* for configuration.

Configure Logging

```
<logging>
    <enable>1</enable>
    <level>5</level>
    <dir>/var/log/zebra-scanner/corescannerd/</dir>
    <max_lines_for_a_file>20000</max_lines_for_a_file>
    <max_number_of_files>200</max_number_of_files>
</logging>
```

- <enable>
  - 1 - Enable logging
  - 0 - Disable logging
- <level>
  - Desiderd level of logging ranges from 1 to 5. Default value is 5.
    NOTE: All log entries will be in level 5 now.

- < dir >
  - Pre-existing location for log creation. The default location is /var/log/zebra-scanner/corescannerd.
- <max_lines_for_a_file>
  - Maximum number of log entries for single log file. Any positive value is valid for this entry.
- <max_number_of_files>
  - Maximum number of log files ranges from 1 to 999. Oldest file will be replaced by a new file once the limit reaches.

Configure Udev Options

```
<udev-options>
    <libudev_library_dir></libudev_library_dir>
</udev-options>
```

- <libudev_library_dir>
  - Custom udev library path

Configure Client Library Options

```
<client-library>
    <binary_data_event>1</binary_data_event>
</client-library>
```

- <binary_data_event>
  - 1 - Enable Binary Data Event; IDC data to be received in ISO1543 format
  - 0 - Disable Binary Data Event; IDC data to be received in separate image and barcode events

CHAPTER 3 SCANNER SDK API

Overview

The Zebra Scanner SDK provides an easy to use and powerful set of API commands to interface with scanners. Commands include:

- Open
- GetScanners
- ExecCommand
- ExecCommandAsync
- Close.

Once you invoke the *Open* and *GetScanners* commands and retrieve the list of connected scanners, all other methods execute using the *ExecCommand* and *ExecCommandAsync* commands. This user friendly approach facilitates coding.

As SDK capabilities grow, the number of methods increases rather than the number of API commands. Once the system is up and running, a new method is just an additional operation to the existing code.

The Zebra Scanner SDK supports seven types of events:

- ImageEvent
- VideoEvent
- BarcodeEvent
- PNPEvent
- ScanRMDEvent
- CommandResponseEvent
- IOEvent

See *Chapter 5, SAMPLE SOURCE CODE* for an example of an application illustrating the Zebra Scanner SDK API.

NOTE For a list of a scanner's supported attribute (parameter) numbers and definitions, refer to the scanner Product Reference Guide, available on the Zebra Support website: http://www.zebra.com/support. Attributes include configuration parameters, monitored data, and asset tracking information.

Scanner ID

The scanner ID uniquely identifies a scanner connected to the CoreScanner driver, and is required to communicate with the device. Developers call the *GetScanners* method to retrieve the scanner IDs of connected devices. For example, to switch on a scanner's red LED, obtain the scanner ID of that scanner and provide this value in the <scannerID> element of inXML in the *ExecCommand* method call.

Each CoreScanner driver instance assigns scanner IDs sequentially to each connected device. Restarting the CoreScanner service re-initializes the array of connected scanners, and previous scanner IDs may no longer be valid. In this case, execute the *GetScanners* method to obtain the new scanner IDs.

During a single CoreScanner driver instance, a Remote Scanner Management (RSM)-supported scanner that is disconnected and reconnected retains its unique scanner ID, while non-RSM device is assigned a different scanner ID each time it is reconnected.

API Commands

Open

Opens an application instance from the user application or user library. This must be the first API command called before invoking any other API command from the user level application.

Syntax

```
unsigned short Open(
    /* [in] */ IEventListenerXml* pEventListener,
    /* [in] */ unsigned int scannerTypeFlags,
    /* [out]*/ StatusID *status);
```

Parameters

- *pEventListener* - Input argument of IEventListener class type object reference.
- *scannerTypeFlags* - The types of scanners requested for use with the API. Use "|" to specify multiple types.

Table 3-1 *Values for scannerTypeFlags*

Code	Value	Scanner Category
SCANNER_TYPES_ALL	0	All Scanners
SCANNER_TYPES_SNAPI	1	SNAPI Scanners
SCANNER_TYPES_IBMHID	2	IBM Hand-held Scanners
SCANNER_TYPES_IBMTT	4	IBM Table-top Scanners
SCANNER_TYPES_HIDKB	8	USB HID Keyboard scanners

Return Values

0 - Success.

Any other value. See *Error and Status Codes on page 3-37*.

GetScanners

Gets a list of connected scanners of the requested types. Invoke this command after the *Open* command.

Syntax

```
unsigned short GetScanners(
    /* [out] */ unsigned short *numberOfScanners,
    /* [out][in] */ std::vector<unsigned int> *scannerIDList,
    /* [out] */ std::string& outXML,
    /* [out] */ StatusID *status);
```

Parameters

- *numberOfScanners* - Number of connected scanners of requested type(s).
- *sfScannerIDList* - Vector/Array of scannerIDs of the requested type(s). The size of the array is 255 (MAX_NUM_DEVICES).
- *outXML* - XML string scanner meta information. See *Chapter 4, TEST UTILITIES & SOURCE CODE* for examples.
- *status* - Return value for the command.

Return Values

0 - Success.

Any other value. See *Error and Status Codes on page 3-37*.

ExecCommand

Provides synchronous execution of a method via an opcode.

Syntax

```
unsigned short ExecCommand(
/* [in] */ unsigned int opcode,
/* [in] */const std::string inXML,
/* [out] */ std::string& outXML,
             /* [out] */ StatusID *status);
```

Parameters

- *opcode* - Method to execute. See *Table 3-11 on page 3-16* for opcodes.
- *inXML* - Relevant argument list for the opcode, structured into an XML string.
- *outXML* - XML string, scanner meta information.
- *status* - Return value for the command.

Return Values

0 - Success.

Any other value. See *Error and Status Codes on page 3-37*.

ExecCommandAsync

Provides asynchronous execution of a method via an opcode. Any response data is retrieved as a *ScannerNotificationEvent*.

Syntax

```
unsigned short ExecCommandAsync(
    /* [in] */ unsigned int opcode,
    /* [in] */ const std::string inXML,
    /* [out] */ StatusID *status);
```

Parameters

- *opcode* - Method to execute. See *Table 3-11 on page 3-16* for opcodes.
- *inXML* - Relevant argument list for the opcode, structured into an XML string.
- *status* - Return value for the command.

Return Values

0 - Success.

Any other value. See *Error and Status Codes on page 3-37*.

Close

Closes the application instance through the CoreScanner service.

Syntax

```
unsigned short Close(
    /* [in] */ unsigned int appHandle,
    /* [out] */ StatusID *status);
```

Parameters

- *appHandle* - Reserved argument. Set to 0.
- *status* - Return value for the command.

Return Values

0 - Success.

Any other value. See *Error and Status Codes on page 3-37*.

API Events

The user application must register for each event category separately to receive events for that category. Use the methods *REGISTER_FOR_EVENTS* and *UNREGISTER_FOR_EVENTS*. See *Table 3-11 on page 3-16*.

ImageEvent

Triggered when an imaging scanner captures images in image mode. To receive ImageEvents, an application must execute the *REGISTER_FOR_EVENTS* method with the SUBSCRIBE_IMAGE event type.

Syntax

```
void OnImageEvent( short eventType,
          int size,
          short imageFormat,
          char* sfimageData,
          int dataLength,
          std::string& pScannerData )
```

Parameters

- *eventType* - Type of image event received. See *Table 3-2*.

Table 3-2 *Image Event Types*

Event Type	Value	Description
IMAGE_COMPLETE	1	A complete image is captured
IMAGE_TRAN_STATUS	2	Image error or status

- *size* - Size of image data buffer.
- *dataLength* - Length of image data buffer.
- *imageFormat* - Format of image. See *Table 3-3*.

Table 3-3 *Image Formats*

Image Type	Value
BMP_FILE_SELECTION	3
TIFF_FILE_SELECTION	4
JPEG_FILE_SELECTION	1

- *sfimageData* - Image data buffer.
- *pScannerData* - Information in XML format about the scanner (ID, Model Number, Serial Number, and GUID) that triggered the image event.

```
<?xml version="1.0" encoding="UTF-8"?>
<outArgs>
    <scannerID>1</scannerID>
    <arg-xml>
        <modelnumber>DS6707-SR20001ZZR</modelnumber>
        <serialnumber>7114000503322</serialnumber>
        <GUID>33C01F39EB23D949B5F3DBF643304FC4</GUID>
    </arg-xml>
</outArgs>
```

VideoEvent

Triggered when an imaging scanner captures video in video mode. To receive VideoEvents, an application must execute the *REGISTER_FOR_EVENTS* method with the SUBSCRIBE_VIDEO event type.

Syntax

```
void OnVideoEvent( short eventType,
                   int size,
                   char* sfvideoData,
                   int dataLength,
                   std::string& pScannerData )
```

Parameters

- *eventType* - Type of video event received. See *Table 3-4*.

Table 3-4 *Video Event Types*

Event Type	Value	Description
VIDEO_FRAME_COMPLETE	1	A complete video frame is captured.

- *size* - Size of video data buffer.
- *sfvideoData* - Video data buffer.
- *dataLength* - Length of video data buffer.
- *pScannerData* - Reserved parameter. Always returns an empty string.

BarcodeEvent

Triggered when a scanner captures barcodes. To receive BarcodeEvents, an application must execute the *REGISTER_FOR_EVENTS* method with the SUBSCRIBE_BARCODE event type.

Syntax

```
void OnBarcodeEvent(short int eventType,
              std::string & pscanData);
```

Parameters

- *eventType* - Type of barcode event received. See *Table 3-5*.

Table 3-5 *Barcode Event Types*

Event Type	Value	Description
SCANNER_DECODE_GOOD	1	A decode is successful.

- *pscanData* - Barcode string that contains information about the scanner that triggered the barcode event including data type, data label, and raw data of the scanned barcode.

```
<?xml version="1.0" encoding="UTF-8"?>
<outArgs>
   <scannerID>1</scannerID>
   <arg-xml>
      <scandata>
         <modelnumber>DS6707-SR20001ZZR</modelnumber>
         <serialnumber>7114000503322</serialnumber>
         <GUID>33C01F39EB23D949B5F3DBF643304FC4</GUID>
         <datatype>8</datatype>
         <datalabel>0x30 0x32 0x31 0x38 0x39 0x38 0x36 0x32</datalabel>
         <rawdata>0x30 0x32 0x31 0x38 0x39 0x38 0x36 0x32</rawdata>
      </scandata>
   </arg-xml>
</outArgs>
```

<datatype> indicates the barcode type of the scanned barcode.

Table 3-6 lists the values received in SNAPI and IBM Hand-held USB communication protocols for each supported barcode type.

Table 3-6 *Barcode Data Types*

Barcode Data Type	Communication Protocol	
	SNAPI	IBM Hand-held
Code 39	1	1
Codabar	2	2
Code 128	3	3
Discrete (Standard) 2 of 5	4	4
IATA	5	N/A

N/A indicates that the communication protocol does not support that barcode data type. The SDK typically returns a value of 0 for these data types, but in some cases may identify these as a related data type. For example, Nixdorf Mode B does not support UPC-A + 2 Supplemental but the SDK identifies it as UPC-A.

Table 3-6 *Barcode Data Types (Continued)*

Barcode Data Type	Communication Protocol	
	SNAPI	IBM Hand-held
Interleaved 2 of 5	6	6
Code 93	7	7
UPC-A	8	8
UPC-E0	9	9
EAN-8	10	10
EAN-13	11	11
Code 11	12	N/A
Code 49	13	13
MSI	14	N/A
EAN-128	15	15
UPC-E1	16	N/A
PDF-417	17	17
Code 16K	18	N/A
Code 39 Full ASCII	19	N/A
UPC-D	20	N/A
Code 39 Trioptic	21	N/A
Bookland	22	N/A
Coupon Code	23	N/A
NW-7	24	N/A
ISBT-128	25	N/A
Micro PDF	26	N/A
DataMatrix	27	27
QR Code	28	28
Micro PDF CCA	29	N/A
PostNet US	30	N/A
Planet Code	31	N/A
Code 32	32	N/A
ISBT-128 Con	33	N/A

N/A indicates that the communication protocol does not support that barcode data type. The SDK typically returns a value of 0 for these data types, but in some cases may identify these as a related data type. For example, Nixdorf Mode B does not support UPC-A + 2 Supplemental but the SDK identifies it as UPC-A.

Table 3-6 *Barcode Data Types (Continued)*

Barcode Data Type	Communication Protocol	
	SNAPI	IBM Hand-held
Japan Postal	34	N/A
Australian Postal	35	N/A
Dutch Postal	36	N/A
MaxiCode	37	37
Canadian Postal	38	N/A
UK Postal	39	N/A
Macro PDF	40	N/A
Micro QR code	44	44
Aztec	45	45
GS1 Databar (RSS-14)	48	48
RSS Limited	49	49
GS1 Databar Expanded (RSS Expanded)	50	50
Scanlet	55	N/A
UPC-A + 2 Supplemental	72	72
UPC-E0 + 2 Supplemental	73	73
EAN-8 + 2 Supplemental	74	74
EAN-13 + 2 Supplemental	75	75
UPC-E1 + 2 Supplemental	80	N/A
CCA EAN-128	81	N/A
CCA EAN-13	82	N/A
CCA EAN-8	83	N/A
CCA RSS Expanded	84	N/A
CCA RSS Limited	85	N/A
CCA RSS-14	86	N/A
CCA UPC-A	87	N/A
CCA UPC-E	88	N/A
CCC EAN-128	89	N/A
TLC-39	90	N/A

N/A indicates that the communication protocol does not support that barcode data type. The SDK typically returns a value of 0 for these data types, but in some cases may identify these as a related data type. For example, Nixdorf Mode B does not support UPC-A + 2 Supplemental but the SDK identifies it as UPC-A.

Table 3-6 *Barcode Data Types (Continued)*

Barcode Data Type	Communication Protocol	
	SNAPI	IBM Hand-held
CCB EAN-128	97	N/A
CCB EAN-13	98	N/A
CCB EAN-8	99	N/A
CCB RSS Expanded	100	N/A
CCB RSS Limited	101	N/A
CCB RSS-14	102	N/A
CCB UPC-A	103	N/A
CCB UPC-E	104	N/A
Signature Capture	105	N/A
Matrix 2 of 5	113	N/A
Chinese 2 of 5	114	N/A
UPC-A + 5 Supplemental	136	136
UPC-E0 + 5 Supplemental	137	137
EAN-8 + 5 Supplemental	138	138
EAN-13 + 5 Supplemental	139	139
UPC-E1 + 5 Supplemental	144	N/A
Macro Micro PDF	154	N/A

N/A indicates that the communication protocol does not support that barcode data type. The SDK typically returns a value of 0 for these data types, but in some cases may identify these as a related data type. For example, Nixdorf Mode B does not support UPC-A + 2 Supplemental but the SDK identifies it as UPC-A.

PNPEvent

Triggered when a scanner of a requested type attaches to or detaches from the system. Pairing a Bluetooth scanner to a cradle does not trigger a PnP event. To receive information about a newly paired device, call the *GetScanners* command again.

To receive PnPEvents, an application must execute the *REGISTER_FOR_EVENTS* method with the SUBSCRIBE_PNP event type.

Syntax

```
void OnPNPEvent( short eventType,
                 std::string ppnpData )
```

Parameters

- *eventType* - Type of PnP event received. See *Table 3-7*.

Table 3-7 *PnP Event Types*

Event Type	Value	Description
SCANNER_ATTACHED	0	A Zebra scanner is attached.
SCANNER_DETACHED	1	A Zebra scanner is detached.

- *ppnpData* - PnP information string containing the asset tracking information of the attached or detached device.

Samples

Sample ppnpData XML for attachment of a direct scanner:

```
<?xml version="1.0" encoding="UTF-8"?>
<outArgs>
  <arg-xml>
    <scanners>
      <scanner type="SNAPI">
        <scannerID>1</scannerID>
        <modelnumber>DS9808-SR00007C1WR</modelnumber>
        <serialnumber>1026300507698   </serialnumber>
        <GUID>77E48FC31C75444B90BE318FECFAE867</GUID>
      </scanner>
    </scanners>
    <status>1</status>
  </arg-xml>
</outArgs>
```

Sample ppnpData XML for attachment of a cascaded scanner. This XML can be received as a PnP event after a *GetScanners* command if there are devices newly paired to a Bluetooth cradle.

```
<?xml version="1.0" encoding="UTF-8"?>
<outArgs>
  <arg-xml>
    <scanners>
      <scanner type="USBIBMHID">
        <scannerID>1</scannerID>
        <modelnumber>CR0078-SC10007WR  </modelnumber>
        <serialnumber>1020800512980   </serialnumber>
        <GUID>3665579766A9514DAAF523D35E051674</GUID>
        <pnp>0</pnp>
        <scanner type="USBIBMHID">
          <scannerID>2</scannerID>
          <modelnumber>DS6878-SR20007WR  </modelnumber>
          <serialnumber>M1M87R38Y       </serialnumber>
          <GUID></GUID>
          <pnp>1</pnp>
        </scanner>
      </scanner>
    </scanners>
    <status>1</status>
  </arg-xml>
</outArgs>
```

Information about Bluetooth Cradle

Information about Bluetooth Scanner

ScanRMDEvent

Receives RMD events when updating scanner firmware. To receive RMD events, an application must execute the *REGISTER_FOR_EVENTS* method with the SUBSCRIBE_RMD event type.

Syntax

```
void OnScanRMDEvent ( short eventType,
                      std::string & pscanData)
```

Parameters

- *eventType* - Type of RMD event received. See *Table 3-8*.

Table 3-8 *RMD Event Types*

Event Type	Value	Description
SCANNER_UF_SESS_START	11	Flash download session started.
SCANNER_UF_DL_START	12	Component download started.
SCANNER_UF_DL_PROGRESS	13	Block(s) of flash completed.
SCANNER_UF_DL_END	14	Component download ended.
SCANNER_UF_SESS_END	15	Flash download session ended.
SCANNER_UF_STATUS	16	Updated error or status.

- *prmdData* - ScanRMD information string containing the event data. See *Firmware Upgrade Scenarios on page 4-26* for more information.

ScannerNotificationEvent

Received when a SNAPI scanner changes operational mode. To receive ScannerNotificationEvents, an application must execute the *REGISTER_FOR_EVENTS* method with the SUBSCRIBE_OTHER event type.

Syntax

```
void OnScannerNotification ( short notificationType,
                 std::string & pScannerData);
```

Parameters

- *notificationType* - Type of the notification event received. See *Table 3-9*.

Table 3-9 *Notification Event Types*

Notification Type	Value	Description
DECODE_MODE	1	Triggered when a scanner changes its operation mode to decode.
SNAPSHOT_MODE	2	Triggered when a scanner changes its operation mode to image mode.
VIDEO_MODE	3	Triggered when a scanner changes its operation mode to video mode.

- *pScannerData* - Information about the scanner (ID, Model Number, Serial Number and GUID) that triggered the notification event.

BinaryDataEvent

Triggered when an IDC-supported imaging scanner captures an image in Intelligent Document Capture (IDC). To receive a BinaryDataEvent, an application must execute the *REGISTER_FOR_EVENTS* method with the SUBSCRIBE_IMAGE event type. See *Configure Client Library Options on page 2-11* for more information.

Syntax

```
void OnBinaryDataEvent( short eventType, int size, short dataFormat,
unsigned char* sfBinaryData, std::string&  pScannerData);
```

Parameters

- *eventType* - Reserved
- *size* - Size of binary data buffer
- *dataFormat* - The format of the BinaryDataEvent

Table 3-10 *Binary data Formats*

Data format	Data format
0xB5	IDC format

- *sfBinaryData* - Binary data buffer.
- *pScannerData* - Information in XML format about the scanner (ID, Model Number, Serial Number, and GUID) that triggered the Binary data event.

```
<?xml version="1.0" encoding="UTF-8"?>
<outArgs>
    <scannerID>1</scannerID>
    <arg-xml>
        <modelnumber>DS6707-SR20001ZZR</modelnumber>
        <serialnumber>7114000503322</serialnumber>
        <GUID>33C01F39EB23D949B5F3DBF643304FC4</GUID>
    </arg-xml>
</outArgs>
```

Methods Invoked Through ExecCommand or ExecCommandAsync

Table 3-11 *List of Methods*

Description	Method	Value	Page
Scanner SDK Commands	GET_VERSION	1000	
	REGISTER_FOR_EVENTS	1001	*3-20*
	UNREGISTER_FOR_EVENTS	1002	*3-21*
Scanner Access Control Commands	CLAIM_DEVICE	1500	*3-21*
	RELEASE_DEVICE	1501	*3-21*
Scanner Common Commands	ABORT_MACROPDF	2000	*3-22*
	ABORT_UPDATE_FIRMWARE	2001	*3-22*
	AIM_OFF	2002	*3-22*
	AIM_ON	2003	*3-23*
	FLUSH_MACROPDF	2005	*3-23*
	DEVICE_PULL_TRIGGER	2011	*3-23*
	DEVICE_RELEASE_TRIGGER	2012	*3-24*
	SCAN_DISABLE	2013	*3-24*
	SCAN_ENABLE	2014	*3-24*
	REBOOT_SCANNER	2019	*3-25*
Scanner Operation Mode Commands	DEVICE_CAPTURE_IMAGE	3000	*3-25*
	DEVICE_CAPTURE_BARCODE	3500	*3-25*
	DEVICE_CAPTURE_VIDEO	4000	*3-26*
Scanner Management Commands	ATTR_GETALL	5000	*3-27*
	ATTR_GET	5001	*3-29*
	ATTR_GETNEXT	5002	*3-31*
	ATTR_SET	5004	*3-32*
	ATTR_STORE	5005	*3-32*
	GET_DEVICE_TOPOLOGY	5006	*3-33*
	START_NEW_FIRMWARE	5014	*3-33*
	UPDATE_FIRMWARE	5016	*3-34*
	UPDATE_FIRMWARE_FROM_PLUGIN	5017	*3-34*

*** Values for the SET_ACTION method are available in Table 3-12.**

Table 3-11 *List of Methods (Continued)*

Description	Method	Value	Page
Scanner Action Commands *	SET_ACTION	6000	*3-35*
Other Commands	DEVICE_SWITCH_HOST_MODE	6200	*3-36*
*** Values for the SET_ACTION method are available in Table 3-12.**			

Table 3-12 *Action Attributes and Values*

Attribute Number (Opcode)	Attribute Name	Description	Data Type	Values
6000	Beeper/LED	Triggers the beeper/LED via command	'X'	0 - 1 high short beep 1 - 2 high short beeps 2 - 3 high short beeps 3 - 4 high short beeps 4 - 5 high short beeps 5 - 1 low short beep 6 - 2 low short beeps 7 - 3 low short beeps 8 - 4 low short beeps 9 - 5 low short beeps 10 - 1 high long beep 11 - 2 high long beeps 12 - 3 high long beeps 13 - 4 high long beeps 14 - 5 high long beeps 15 - 1 low long beep 16 - 2 low long beeps 17 - 3 low long beeps 18 - 4 low long beeps 19 - 5 low long beeps 20 - Fast warble beep 21 - Slow warble beep 22 - High-low beep 23 - Low-high beep 24 - High-low-high beep 25 - Low-high-low beep 26 - High-high-low-low beep 42 - Green LED off 43 - Green LED on 45 - Yellow LED 46 - Yellow LED off 47 - Red LED on 48 - Red LED off
6001	ParameterDefaults	Initiates a parameter defaults command	'X'	0 - Restore defaults 1 - Restore factory defaults 2 - Write custom defaults
6003	BeepOnNextBootup	Controls whether to suppress boot up / power up beep on the next power up	'X'	0 - Disable beep on next bootup 1 - Enable beep on next bootup
6004	Reboot	Remote reboot command	'X'	

Table 3-12 *Action Attributes and Values (Continued)*

Attribute Number (Opcode)	Attribute Name	Description	Data Type	Values
6005	HostTriggerSession	Triggers the scanner to start scanning via command	'X'	0 - Start host trigger session 1 - Stop host trigger session
6011	StatsReset	Reset/default a specific statistic	'X'	The specific statistic attribute to reset. Range: 15002-19999
6013	StatsResetAll	Reset/default all statistics	'X'	'
6017	ScaleReadWeight	Read weight from scale	'A'	Byte[0] status: 0 - scaleNotEnabled 1 - scaleNotReady 2 - stableWeightOverLimit 3 - stableWeightUnderZero 4 - nonStableWeight 5 - stableZeroWeight 6 - stableNonZeroWeight Byte[1] units: 0=kgs 1=lbs Bytes[2-5] weight in thousandths of units
6018	ScaleZero	Zeros the scale	'X'	
6019	ScaleReset	Resets the scale	'X'	
6022	ChangeAllCodeTypes	Enables/Disables all code types	'X'	0 = Disable all code types 1 = Enable all code types

Method Examples

The following inXML segments are examples only. Program inXML strings for customization based on application requirements.

REGISTER_FOR_EVENTS

Value 1001

Description: Register for API events described in *API Events beginning on page 3-6*

Asynchronous supported: No

Supported scanner communication protocols: N/A

inXml:

```
<inArgs>
  <cmdArgs>
    <arg-int>6</arg-int>
    <arg-int>1,2,4,8,16,32</arg-int>
  </cmdArgs>
</inArgs>
```

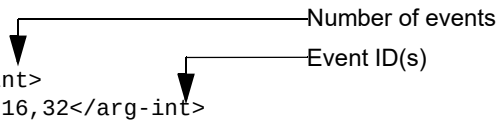


Table 3-13 lists the Event IDs for the inXML code above.

Table 3-13 *Event IDs*

Event Name	Event ID
SUBSCRIBE_BARCODE	1
SUBSCRIBE_IMAGE	2
SUBSCRIBE_VIDEO	4
SUBSCRIBE_RMD	8
SUBSCRIBE_PNP	16
SUBSCRIBE_OTHER	32

outXml: null

UNREGISTER_FOR_EVENTS

Value 1002

Description: Unregister from API events described in *API Events beginning on page 3-6*

Asynchronous supported: No

Supported scanner communication protocols: N/A

inXml:

```
<inArgs>
  <cmdArgs>
    <arg-int>6</arg-int>
    <arg-int>1,2,4,8,16,32</arg-int>
  </cmdArgs>
</inArgs>
```

Number of Events

Event ID(s)

outXml: null

CLAIM_DEVICE

Value 1500

Description: Claim a specified device

Asynchronous supported: No

Supported scanner communication protocols: N/A

inXml:

```
<inArgs>
  <scannerID>1</scannerID>
</inArgs>
```

Specified Scanner ID

outXml: null

RELEASE_DEVICE

Value 1501

Description: Release a specified device

Asynchronous supported: No

Supported scanner communication protocols: N/A

inXml:

```
<inArgs>
  <scannerID>1</scannerID>
</inArgs>
```

Specified Scanner ID

outXml: null

ABORT_MACROPDF

Value 2000

Description: Abort MacroPDF of a specified scanner

Asynchronous supported: No

Supported scanner communication protocols: SNAPI

inXml:

```
<inArgs>
  <scannerID>1</scannerID>
</inArgs>
```

Specified Scanner ID

outXml: null

ABORT_UPDATE_FIRMWARE

Value 2001

Description: Abort firmware update process of a specified scanner

Asynchronous supported: No

Supported scanner communication protocols: IBM Hand-held, IBM Table-top, SNAPI

inXml:

```
<inArgs>
  <scannerID>1</scannerID>
</inArgs>
```

Specified Scanner ID

outXml: null

AIM_OFF

Value 2002

Description: Turn off the aiming of a specified scanner

Asynchronous supported: No

Supported scanner communication protocols: SNAPI

inXml:

```
<inArgs>
  <scannerID>1</scannerID>
</inArgs>
```

Specified Scanner ID

outXml: null

AIM_ON

Value 2003

Description: Turn on the aiming of a specified scanner

Asynchronous supported: No

Supported scanner communication protocols: SNAPI

inXml:

```
<inArgs>
  <scannerID>1</scannerID>
</inArgs>
```

Specified Scanner ID

outXml: null

FLUSH_MACROPDF

Value 2005

Description: Flush MacroPDF of a specified scanner

Asynchronous supported: No

Supported scanner communication protocols: SNAPI

inXml:

```
<inArgs>
  <scannerID>1</scannerID>
</inArgs>
```

Specified Scanner ID

outXml: null

DEVICE_PULL_TRIGGER

Value 2011

Description: Pull the trigger of a specified scanner

Asynchronous supported: N/A

Supported scanner communication protocols: SNAPI, IBM Hand-held*, IBM Table-top

* Supported auxiliary scanners if the firmware supports.

inXml:

```
<inArgs>
  <scannerID>1</scannerID>
</inArgs>
```

Specified Scanner ID

outXml: null

DEVICE_RELEASE_TRIGGER

Value 2012

Description: Release the pulled trigger of a specified scanner

Asynchronous supported: N/A

Supported scanner communication protocols: SNAPI, IBM Hand-held*, IBM Table-top*

* Supported auxiliary scanners if the firmware supports.

inXml:

```
<inArgs>
  <scannerID>1</scannerID>
</inArgs>
```

Specified Scanner ID

outXml: null

SCAN_DISABLE

Value 2013

Description: Disable scanning on a specified scanner

Asynchronous supported: N/A

Supported scanner communication protocols: SNAPI, IBM Hand-held, IBM Table-top

inXml:

```
<inArgs>
  <scannerID>1</scannerID>
</inArgs>
```

Specified Scanner ID

outXml: null

SCAN_ENABLE

Value 2014

Description: Enable scanning on a specified scanner

Asynchronous supported: N/A

Supported scanner communication protocols: SNAPI, IBM Hand-held, IBM Table-top

inXml:

```
<inArgs>
  <scannerID>1</scannerID>
</inArgs>
```

Specified Scanner ID

outXml: null

REBOOT_SCANNER

Value 2019

Description: Reboot a scanner. To reboot a Bluetooth scanner, send this command to the scanner's associated cradle.

Asynchronous supported: N/A

Supported scanner communication protocols: IBM Hand-held, IBM Table-top, SNAPI

inXml:

```
<inArgs>
  <scannerID>1</scannerID>
</inArgs>
```

Specified Scanner ID

outXml: null

DEVICE_CAPTURE_IMAGE

Value 3000

Description: Invoke snapshot mode on an imaging scanner. The scanner blinks the green LED at one second intervals to indicate it is not in standard operating (decode) mode, and reverts to standard operating mode after a trigger pull or the snapshot time out occurs. After a trigger pull, the CoreScanner driver triggers an ImageEvent containing the captured image. See *ImageEvent on page 3-6*.

Asynchronous supported: N/A

Supported scanner communication protocols: SNAPI

inXml:

```
<inArgs>
  <scannerID>1</scannerID>
</inArgs>
```

Specified Scanner ID

outXml: null

DEVICE_CAPTURE_BARCODE

Value 3500

Description: Change a scanner to decode mode.

Asynchronous supported: N/A

Supported scanner communication protocols: SNAPI

inXml:

```
<inArgs>
  <scannerID>1</scannerID>
</inArgs>
```

Specified Scanner ID

outXml: null

DEVICE_CAPTURE_VIDEO

Value 4000

Description:	Invoke video mode on an imaging scanner. The scanner behaves as a video camera as long as the trigger is pulled. Releasing the trigger returns the scanner to decode mode. As long as the trigger is pulled, the CoreScanner driver triggers VideoEvents that contain the video data. See *VideoEvent on page 3-7*.
Asynchronous supported:	N/A
Supported scanner communication protocols:	SNAPI

inXml:

```
<inArgs>
  <scannerID>1</scannerID>
</inArgs>
```

outXml: null

ATTR_GETALL

Value 5000

Description: Get all attributes of a scanner. A synchronous call of this method returns an outXML as in the following example. An asynchronous call triggers a CommandResponseEvent. See *page 3-14*.

Asynchronous supported: Yes

Supported scanner communication protocols: IBM Hand-held, IBM Table-top, SNAPI

inXml:

```
<inArgs>
  <scannerID>1</scannerID>
</inArgs>
outXml:
<?xml version="1.0" encoding="UTF-8" ?>
<outArgs>
  <scannerID>1</scannerID>
  <arg-xml>
    <modelnumber>DS670-SR20001ZZR</modelnumber>
    <serialnumber>7116000501003</serialnumber>
    <GUID>A2E647DED2163545B18BCEBD0A2A133D</GUID>
<response>
      <opcode>5000</opcode>
      <attrib_list>
<attribute name="">0</attribute>
<attribute name="">1</attribute>
<attribute name="">2</attribute>
<attribute name="">3</attribute>
<attribute name="">4</attribute>
<attribute name="">5</attribute>
<attribute name="">6</attribute>
<attribute name="">7</attribute>
<attribute name="">8</attribute>
<attribute name="">9</attribute>
<attribute name="">10</attribute>
<attribute name="">11</attribute>
<attribute name="">12</attribute>
<attribute name="">13</attribute>
<attribute name="">14</attribute>
<attribute name="">15</attribute>
<attribute name="">16</attribute>
```

Specified Scanner ID

Scanner ID of Data Receiving

Asset Tracking Information of the Scanner

Method Response Received

Attribute Numbers

inXml (continued):

```
<attribute name="">17</attribute>
<attribute name="">18</attribute>
<attribute name="">19</attribute>
<attribute name="">20</attribute>
<attribute name="">21</attribute>
<attribute name="">22</attribute>
<attribute name="">23</attribute>
<attribute name="">24</attribute>
<attribute name="">25</attribute>
<attribute name="">26</attribute>
<attribute name="">27</attribute>
<attribute name="">28</attribute>
<attribute name="">29</attribute>
<attribute name="">30</attribute>
<attribute name="">31</attribute>
<attribute name="">34</attribute>
<attribute name="">35</attribute>
<attribute name="">36</attribute>
<attribute name="">37</attribute>
<attribute name="">38</attribute>
<attribute name="">39</attribute>
<attribute name="">655</attribute>
<attribute name="">656</attribute>
<attribute name="">657</attribute>
<attribute name="">658</attribute>
<attribute name="">659</attribute>
<attribute name="">665</attribute>
<attribute name="">670</attribute>
<attribute name="">672</attribute>
<attribute name="">673</attribute>
<attribute name="">705</attribute>
<attribute name="">716</attribute>
<attribute name="">718</attribute>
<attribute name="">721</attribute>
<attribute name="">724</attribute>
<attribute name="">726</attribute>
<attribute name="">727</attribute>
<attribute name="">728</attribute>
<attribute name="">730</attribute>
<attribute name="">731</attribute>
<attribute name="">734</attribute>
<attribute name="">735</attribute>
<attribute name="">745</attribute>
<attribute name="">6000</attribute>
<attribute name="">6001</attribute>
<attribute name="">6002</attribute>
<attribute name="">6003</attribute>
<attribute name="">6004</attribute>
<attribute name="">20004</attribute>
<attribute name="">20006</attribute>
<attribute name="">20007</attribute>
<attribute name="">20008</attribute>
<attribute name="">20009</attribute>
<attribute name="">20010</attribute>
<attribute name="">20011</attribute>
<attribute name="">20013</attribute>
      </attrib_list>
    </response>
  </arg-xml>
</outArgs>
```

Attribute Numbers

NOTE For a list of supported attribute (parameter) numbers and definitions, refer to the scanner Product Reference Guide, available on the Zebra Support website: http://www.zebra.com/support. Attributes include configuration parameters, monitored data, and asset tracking information.

ATTR_GET

Value 5001

Description:	Get the attribute values of a scanner. A synchronous call of this method returns outXML as in the following example. An asynchronous call triggers a CommandResponseEvent. See *ScannerNotificationEvent on page 3-14*.
Asynchronous supported:	Yes
Supported scanner communication protocols:	IBM Hand-held, IBM Table-top, SNAPI

inXml:

```
><inArgs>
  <scannerID>1</scannerID>
  <cmdArgs>
    <arg-xml>
      <attrib_list>535,20004,1,140,392</attrib_list>
    </arg-xml>
  </cmdArgs>
</inArgs>
```

Specified Scanner ID

Required Attribute Numbers

outXML:

```
<?xml version="1.0" encoding="UTF-8" ?>
<outArgs>
  <scannerID>1</scannerID>
  <arg-xml>
    <modelnumber>DS670-SR20001ZZR</modelnumber>
    <serialnumber>7116000501003</serialnumber>
    <GUID>A2E647DED2163545B18BCEBD0A2A133D</GUID>
    <response>
      <opcode>5001</opcode>
      <attrib_list>
        <attribute>
          <id>535</id>
          <name></name>
          <datatype>S</datatype>
          <permission>R</permission>
          <value>27APR07</value>
        </attribute>
        <attribute>
          <id>20004</id>
          <name></name>
          <datatype>S</datatype>
          <permission>R</permission>
          <value>DS6707X4</value>
        </attribute>
<attribute>
          <id>1</id>
          <name></name>
<datatype>F</datatype>
          <permission>RWP</permission>
          <value>True</value>
        </attribute>
        <attribute>
          <id>140</id>
          <name></name>
          <datatype>B</datatype>
          <permission>RWP</permission>
          <value>0</value>
        </attribute>

 <attribute>
          <id>392</id>
          <name></name>
          <datatype>A</datatype>
          <permission>RWP</permission>
          <value>0x01 0x00 0x58 0x55 0x41 0x00 0x0b 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00</value>
        </attribute>
      </attrib_list>
    </response>
  </arg-xml>
</outArgs>
```

Scanner ID of Data Receiving

Asset Tracking Information of the Scanner

Method Response Received

Attribute Number

Attribute Data Typer

Permissions of the Attribute

Attribute Value

ATTR_GETNEXT

Value 5002

Description: Get the value of the next attribute to a given attribute of a scanner. A synchronous call of this method returns an outXML as in the following example. An asynchronous call triggers a CommandResponseEvent. See *ScannerNotificationEvent on page 3-14*.

Asynchronous supported: Yes

Supported scanner communication protocols: IBM Hand-held, IBM Table-top, SNAPI

inXml:

```
<inArgs>
  <scannerID>1</scannerID>
  <cmdArgs>
    <arg-xml>
      <attrib_list>14</attrib_list>
    </arg-xml>
  </cmdArgs>
</inArgs>
```

Specified Scanner ID

Attribute Numbers

outXML:

```
<?xml version="1.0" encoding="UTF-8"?>
<outArgs>
  <scannerID>1</scannerID>
  <arg-xml>
    <modelnumber>DS670-SR20001ZZR</modelnumber>
    <serialnumber>7116000501003</serialnumber>
    <GUID>A2E647DED2163545B18BCEBD0A2A133D</GUID>
    <response>
      <opcode>5002</opcode>
      <attrib_list>
        <attribute>
          <id>15</id>
          <name></name>
          <datatype>F</datatype>
          <permission>RWP</permission>
          <value>True</value>
        </attribute>
      </attrib_list>
    </response>
  </arg-xml>
</outArgs>
```

Scanner ID of Data Receiving

Asset Tracking Information of the Scanner

Method Response Received

Attribute Numbers

Permissions of the Attribute

Attribute Value

NOTE If the next available attribute is not readable (for example, an Action attribute), this command returns the next available readable attribute value.

ATTR_SET

Value 5004

Description: Set the attribute values of a scanner. These values are discarded on power down.

Asynchronous supported: N/A

Supported scanner communication protocols: IBM Hand-held, IBM Table-top, SNAPI

inXml:

```
<inArgs>
  <scannerID>1</scannerID>                    <-- Specified Scanner ID
  <cmdArgs>
    <arg-xml>
      <attrib_list>
        <attribute>
          <id>1</id>                          <-- Attribute Numbers
          <datatype>F</datatype>              <-- Attribute Data Type
          <value>False</value>                <-- Attribute Value
        </attribute>
      </attrib_list>
    </arg-xml>
  </cmdArgs>
</inArgs>
```

outXml: null

ATTR_STORE

Value 5005

Description: Store the attribute values of a scanner. These values persist over power down and power up cycles.

Asynchronous supported: N/A

Supported scanner communication protocols: IBM Hand-held, IBM Table-top, SNAPI

inXml:

```
<inArgs>
  <scannerID>1</scannerID>                    <-- Specified Scanner ID
  <cmdArgs>
    <arg-xml>
      <attrib_list>
        <attribute>
          <id>1</id>                          <-- Attribute Number
          <datatype>F</datatype>              <-- Attribute Data Type
          <value>False</value>                <-- Attribute Value
        </attribute>
      </attrib_list>
    </arg-xml>
  </cmdArgs>
```

outXml: null

GET_DEVICE_TOPOLOGY

Value 5006

Description: Get the topology of devices connected to the calling system.

Asynchronous supported: No

Supported scanner communication protocols: IBM Hand-held, IBM Table-top, SNAPI

inXml:

```
<inArgs></inArgs>
```

outXML:

```
<?xml version="1.0" encoding="UTF-8"?>
<outArgs>
  <arg-xml>
    <scanners>
      <scanner type="SNAPI">
        <scannerID>1</scannerID>
        <modelnumber>DS670-SR20001ZZR</modelnumber>
        <serialnumber>7116000501003</serialnumber>
        <GUID>A2E647DED2163545B18BCEBD0A2A133D</GUID>
        <VID>1504</VID>
        <PID>6400</PID>
        <DoM>24MAR10</DoM>
        <firmware>NBRPUAAM</firmware>
      </scanner>
      <scanner type="USBIBMHID">
        <scannerID>2</scannerID>
        <modelnumber>CR0078-SC10007WR</modelnumber>
        <serialnumber>MXA4WD88</serialnumber>
        <GUID>993DF345C3B00E408E8160116AE9A319</GUID>
        <VID>1504</VID>
        <PID>2080</PID>
        <DoM>24MAR10</DoM>
        <firmware>NBCACAK7</firmware>
        <scanner type="USBIBMHID">
          <scannerID>3</scannerID>
          <serialnumber>M1M87R39H</serialnumber>
          <modelnumber>DS6878-SR20007WR</modelnumber>
          <DoM>08OCT10</DoM>
          <firmware>PAAAJS00-002-N25</firmware>
        </scanner>
      </scanner>
    </scanners>
```

Scanner Type

Asset Tracking Information of the Scanner

Cascaded Scanner

START_NEW_FIRMWARE

Value 5014

Description: Start the updated firmware. This reboots the scanner.

Asynchronous supported: N/A

Supported scanner communication protocols: IBM Hand-held, IBM Table-top, SNAPI

inXml:

```
<inArgs>
  <scannerID>1</scannerID>
</inArgs>
```

Specified Scanner ID

outXml: null

UPDATE_FIRMWARE

Value 5016

Description: Update the firmware of a scanner. Specify the bulk firmware update option for faster firmware download in SNAPI mode. If an application registered for ScanRMDEvents, it receives ScanRMDEvents as described *on page 3-13*.

Asynchronous supported: N/A

Supported scanner communication protocols: IBM Hand-held, IBM Table-top, SNAPI

inXml:

```
<inArgs>
  <scannerID>1</scannerID>
  <cmdArgs>
    <arg-string>D:\ScannerFW\DS6707\NBRPUCAM.DAT</arg-string>
    <arg-int>2</arg-int>
  </cmdArgs>
</inArgs>
```

Specified Scanner ID

Path to the DAT File

Bulk Update Option

outXml: null

UPDATE_FIRMWARE_FROM_PLUGIN

Value 5017

Description: Update the firmware of a scanner using a plug-in. Specify the bulk firmware update option for faster firmware download in SNAPI mode. If an application registered for ScanRMDEvents, it receives ScanRMDEvents as described *on page 3-13*.

Asynchronous supported: N/A

Supported scanner communication protocols: IBM Hand-held, IBM Table-top, SNAPI

inXml:

```
<inArgs>
  <scannerID>1</scannerID>
  <cmdArgs>
    <arg-string>D:\ScannerFW\DS9808\DS9808-COMMON SR MODELS-S-018.SCNPLG</arg-string>
    <arg-int>2</arg-int>
  </cmdArgs>
</inArgs>
```

Specified Scanner ID

Path to the Plug-in File

Bulk Update Option

outXml: null

NOTE This command does not verify supported scanner models of the plug-in. It attempts the firmware update with the DAT file extracted from the specified plug-in file regardless of model.

SET_ACTION

Value 6000

NOTE Values for the SET_ACTION method are available in *Table 3-12 on page 3-18*.

Description:	Perform an action involving the scanner's beeper or LEDs.
Asynchronous supported:	N/A
Supported scanner communication protocols:	IBM Hand-held, IBM Table-top, SNAPI

inXml:

```
<inArgs>
  <scannerID>1</scannerID>          <-- Specified Scanner ID
  <cmdArgs>
    <arg-int>0</arg-int>            <-- Scanner Action Commands
   </cmdArgs>
</inArgs>
```

outXml: null

DEVICE_SWITCH_HOST_MODE

Value 6200

Description:	Switch the USB host mode of a scanner. This reboots the scanner. If the scanner is in HID Keyboard mode, the only supported host variants are IBM Hand-held, IBM Table-top, and SNAPI. To configure this as a silent switch (suppressing reboot beeps) and persist the targeted host mode as permanent, set the parameters in the inXML string. For a Bluetooth scanner, send this command to the scanner's associated cradle.
Asynchronous supported:	N/A
Supported scanner communication protocols:	IBM Hand-held, IBM Table-top, SNAPI, HID Keyboard.

inXml:

```
<inArgs>
  <scannerID>1</scannerID>                          <-- Specified Scanner ID
  <cmdArgs>
    <arg-string>XUA-45001-1</arg-string>            <-- String Code for Target Host Variant
    <arg-bool>TRUE</arg-bool>                       <-- Silent Switch Option
    <arg-bool>FALSE</arg-bool>                      <-- Permanent Change Option
  </cmdArgs>
</inArgs>
```

Table 3-14 lists the string codes for USB host variants.

Table 3-14 *USB Host Variants*

Host Variant	String Code
USB-IBMHID	XUA-45001-1
USB-IBMTT	XUA-45001-2
USB-HIDKB	XUA-45001-3
USB-SNAPI with Imaging	XUA-45001-9
USB-SNAPI without Imaging	XUA-45001-10

outXml:null

Error and Status Codes

Table 3-15 *Error and Status Codes*

Error/ Status Code	Value	Description
SUCCESS	0	Generic success
STATUS_LOCKED	10	Device is locked by another application
ERROR_INVALID_APPHANDLE	100	Invalid application handle. Reserved parameter. Value is zero.
ERROR_COMMLIB_UNAVAILABLE	101	Required Comm Lib is unavailable to support the requested Type
ERROR_NULL_BUFFER_POINTER	102	Null buffer pointer
ERROR_INVALID_BUFFER_POINTER	103	Invalid buffer pointer
ERROR_INCORRECT_BUFFER_SIZE	104	Incorrect buffer size
ERROR_DUPLICATE_TYPES	105	Requested Type IDs are duplicated
ERROR_INCORRECT_NUMBER_OF_TYPES	106	Incorrect value for number of Types
ERROR_INVALID_ARG	107	Invalid argument
ERROR_INVALID_SCANNERID	108	Invalid scanner ID
ERROR_INCORRECT_NUMBER_OF_EVENTS	109	Incorrect value for number of Event IDs
ERROR_DUPLICATE_EVENTID	110	Event IDs are duplicated
ERROR_INVALID_EVENTID	111	Invalid value for Event ID
ERROR_DEVICE_UNAVAILABLE	112	Required device is unavailable
ERROR_INVALID_OPCODE	113	Opcode is invalid
ERROR_INVALID_TYPE	114	Invalid value for Type
ERROR_ASYNC_NOT_SUPPORTED	115	Opcode does not support asynchronous method
ERROR_OPCODE_NOT_SUPPORTED	116	Device does not support the Opcode
ERROR_OPERATION_FAILED	117	Operation failed in device
ERROR_REQUEST_FAILED	118	Request failed in CoreScanner
ERROR_OPERATION_NOT_SUPPORTED_FOR_AUXILIARY_SCANNERS	119	Operation not supported for auxiliary scanners
ERROR_DEVICE_BUSY	120	Device busy, retry the command
ERROR_ALREADY_OPENED	200	CoreScanner is already opened
ERROR_ALREADY_CLOSED	201	CoreScanner is already closed
ERROR_CLOSED	202	CoreScanner is closed
ERROR_INVALID_INXML	300	Malformed inXML

Table 3-15 *Error and Status Codes (Continued)*

Error/ Status Code	Value	Description
ERROR_XMLREADER_NOT_CREATED	301	Could not instantiate XML Reader
ERROR_XMLREADER_INPUT_NOT_SET	302	Could not set input for XML Reader
ERROR_XMLREADER_PROPERTY_NOT_SET	303	Could not set XML Reader property
ERROR_XMLWRITER_NOT_CREATED	304	Could not instantiate XML Writer
ERROR_XMLWRITER_OUTPUT_NOT_SET	305	Could not set output for XML Writer
ERROR_XMLWRITER_PROPERTY_NOT_SET	306	Could not set XML Writer property
ERROR_XML_ELEMENT_CANT_READ	307	Cannot read element from XML input
ERROR_XML_INVALID_ARG	308	Arguments in inXML are not valid
ERROR_XML_WRITE_FAIL	309	Write to XML output string failed
ERROR_XML_INXML_EXCEED_LENGTH	310	InXML length exceeded
ERROR_XML_EXCEED_BUFFER_LENGTH	311	Buffer length for type exceeded
ERROR_NULL_POINTER	400	Null pointer
ERROR_DUPLICATE_CLIENT	401	Cannot add a duplicate client
ERROR_FW_INVALID_DATFILE	500	Invalid firmware file
ERROR_FW_UPDATE_FAILED_IN_SCN	501	Firmware update failed in scanner
ERROR_FW_READ_FAILED_DATFILE	502	Failed to read DAT file
ERROR_FW_UPDATE_INPROGRESS	503	Firmware update in progress (cannot proceed another firmware update or another command)
ERROR_FW_UPDATE_ALREADY_ABORTED	504	Firmware update is already aborted
ERROR_FW_UPDATE_ABORTED	505	Firmware update aborted
ERROR_FW_SCN_DETTACHED	506	Scanner is disconnected during firmware update
STATUS_FW_SWCOMP_RESIDENT	600	Software component is already resident in the scanner

CHAPTER 4 TEST UTILITIES & SOURCE CODE

Overview

This chapter provides information for evaluating Zebra Scanner SDK software components using the SDK test utilities.

For a list of popular topics within this guide, see *Quick Startup* in the back of the guide.

NOTE For supported scanner attributes (parameters), refer to the scanner Product Reference Guide. This guide may also contain an appendix listing non-parameter attributes that most Zebra scanners support.

Install Verifier Application (IVA) Utility

The Zebra Scanner SDK versions 4.0 and later provide the Install Verifier Application (IVA) utility, a self-installable source tarball package that verifies the CoreScanner and SDK utilities are installed and working properly.

After installing the CoreScanner and SDK packages on a Linux distribution, the IVA application performs the following checks:

- CoreScanner and SDK package installation status using the standard package manager program or Linux commands.
- File-by-file check on installed files.
- Verify generic actions/behaviors of CoreScanner. Check cscore daemon to ensure systemd or systemv daemon is working properly, and ensure that operations (service controlling commands such like start, stop, restart, and status) on cscore daemon are working properly.
- Ability to test basic CoreScanner commands and events.
- Installation and environment setup for CoreScanner and JPOS test tools.

IVA execution generates a summary report containing the status of these verification tests.

IVA Outcome

The utility executes in a predefined order and generates a text report indicating the status of the test. This report verifies that driver installation is successful and working as expected.

Following is a section of a sample output report:

```
==========================================
Zebra Technologies Corescanner Driver Install Verifier App
============================================================
Install verifier app started:                [08122016_03:32:56]
  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
You are running root.....................................................[  SUCCESS ]
  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
Found RPM based package manager in your system.
Found Debain based package manager in your system.
zebra-scanner-corescanner package.....................................[  FOUND   ]
zebra-scanner-devel package...........................................[  FOUND   ]
zebra-scanner-javapos package.........................................[  FOUND   ]
  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
```

Installing the IVA

To install the IVA utility:

1. Place install-verifier-app.tar.gz into the selected directory using the cp command or the system GUI.
2. Extract install-verifier-app.tar.gz using the tar command or the system GUI.

   $> tar -xvf install-verifier-app.tar.gz
3. Open a terminal window. Change the directory to install-verifier-app and build the source. The zebra-install-verifier executable file is now available in the directory.

   $> make

   g++ -c main.cpp -o main.o -I/usr/share/include/zebra-scanner

   g++ -c pugixml.cpp -o pugixml.o -I/usr/include/zebra-scanner

   g++ -c xml_formatter.cpp -o xml_formatter.o -I/usr/include/zebra-scanner

   g++ -o zebra-install-verifier main.o pugixml.o xml_formatter.o -L/usr/share/zebra-scanner/corescanner/ -lcs-client

Execute the zebra-install-verifier using the command line interface.

Executing the IVA

The auto_run.sh script is available in the same directory as zebra-install-verifier. Ensure you have root permission to execute the script.

```
$> sh auto_run.sh
```

The tool runs initial commands in the background and displays the status on the console/terminal window. Use the self-help menu to choose the test case and the execution steps.

Sample menu from the app:

```
============================================================
==1.: Continue to next test after the PNP event.
==2.: Skip to next test.
==0.: Exit.
============================================================
```

Continue to test and verify CoreScanner installation using this menu. Applications refresh at the end of each verification entry.

IVA verifies the following:

- Status of CoreScanner and SDK installation (RPM/Debian package and source tarball installation)
- CoreScanner daemon is working properly
- CoreScanner command GetScanner
- PNP events
- Barcode scanning events
- CoreScanner Command: RSM Get All Attributes
- CoreScanner Command: RSM Get Attribute
- CoreScanner Command: Execute an action attribute (beep, LED ON/OFF)
- SNAPI Imaging with pull trigger and release trigger
- SNAPI direct commands: AIM ON/OFF

Test Utilities

The Zebra Scanner SDK includes the following test utilities that demonstrate SDK functionality to help you gain an understanding of the Zebra Scanner SDK.

- Zebra Scanner SDK C++ GUI Demonstration Application, GTK
- Zebra Scanner SDK C++ Console Sample Application, Commandline (see *Chapter 5, SAMPLE SOURCE CODE*)

NOTE You may need to install all dependent packages before installing the CoreScanner.

The Zebra Scanner SDK test utilities support the following functionality:

- Discovery of asset tracking information
- Barcode scanning
- Image and video capture
- Attribute query and setting
- Host variant switching
- Firmware upgrade.

Scanner SDK C++ GTK-based Sample Application

The Scanner SDK C++ Sample Application demonstrates SDK functionality by simulating an application that communicates with the Scanner SDK. It includes C++ source code and solution and project files for further reference.

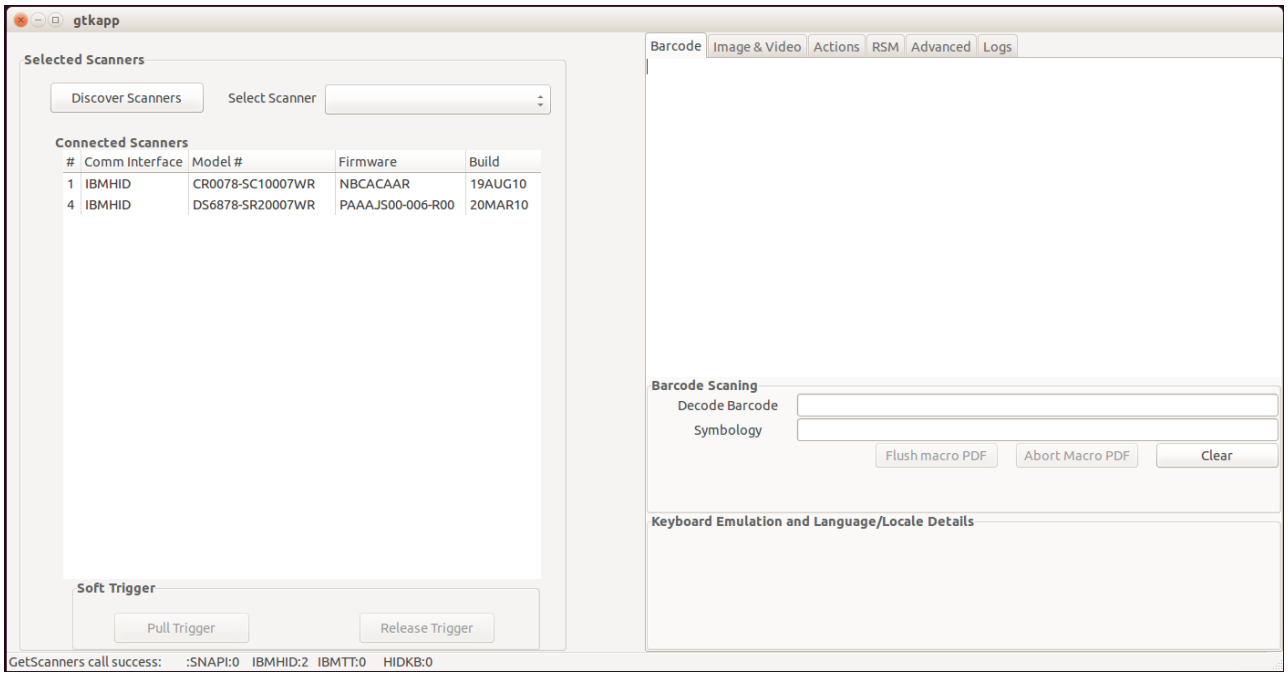


Figure 4-1 *C++ Sample Application*

Table 4-1 *Test Utility Buttons and Fields by Tab*

Button or Field	Description
Discover Scanners	Invoke Open and GetScanners methods and register for all events
Select Scanner	Select the scanner to invoke the command
Connected Scanners	List all connected scanners regardless of mode
Pull Trigger	Soft Pull Trigger the scanner for barcode, image, and video actions
Release Trigger	Soft Release Trigger the scanner for barcode, image, and video actions
Barcode Tab	
Flush Macro PDF	Flush Macro PDF barcode buffer
Abort Macro PDF	Abort Macro PDF read
Clear	Clear the barcode data area
Decoded Barcode	Display the label value of the scanned barcode
Symbology	Display the symbology of the scanned barcode
Image/Video Tab	
Image	Invoke image capture mode
Video	Invoke video capture mode
Abort Transfer	Abort image transfer on serial scanners
Image Type	Select JPG, TIFF, or BMP image type
Enable Video View Finder	Enable the view finder in image mode
Save Image	Save the captured image
Scanner Actions Tab	
Enable/Disable Scanner	Enable/Disable the scanner for data/image/video capture initiation
Aim	Switch on and off Aim control of the scanner
Beeper	Beep the beeper of the scanner
Reboot Scanner	Reboot the scanner
LED	Light the LED(s) on the scanner
Switch Host Variant	Switch the scanner host type; you can select a silent feature and persist the variant change
RSM Tab	
Get All IDs	Get all supported attribute IDs from the selected scanner
Get Value	Select one or more attribute IDs and get their values
Next Value	Get the next attribute value given the current attribute number
Store Value	Store value(s) for selected attribute(s)

Table 4-1 *Test Utility Buttons and Fields by Tab (Continued)*

Button or Field	Description
Set Value	Set value(s) for selected attribute(s)
Select All	Select all attribute IDs at the RSM data viewer
Clear All	Clear all attribute data at the RSM data viewer
Clear All Values	Clear all attribute values at the RSM data viewer (C# only)
Clear Value	Clear a selected attribute value at the RSM data viewer (C# only)
Advance Tab	
Firmware Update Operations	Update firmware and launch the new firmware on the scanner
Browse	Browse the firmware file (*.DAT) or plug-in file (*.SCNPLG)
Update	Initiate firmware update
Abort	Abort firmware update
Launch	Once firmware update finishes launch the new firmware in the scanner
Claim Scanner	Exclusively claim and declaim the scanner for this application
Logs Tab	
Event Log	Command and event log; logs initiated commands
XML log	Displays output of each function if available
Clear Event Log	Clear command and event log area
Clear XML Log	Clear XML log area

Verifying Scanner SDK Functionality

This section provides test cases to demonstrate Zebra Scanner SDK functionality. See *Installing and Configuring CoreScanner and SDK on page 2-5* for more information.

Scanner Discovery / Asset Tracking / Successful SDK Installation Validation

1. Connect a Zebra USB scanner(s) to the host computer.
2. Scan one of the following barcodes to place the scanner in USB Hand-held or USB SNAPI mode.

USB (IBM Hand-held)

USB SNAPI

3. Launch the Zebra Scanner SDK Sample Utility on the host computer.

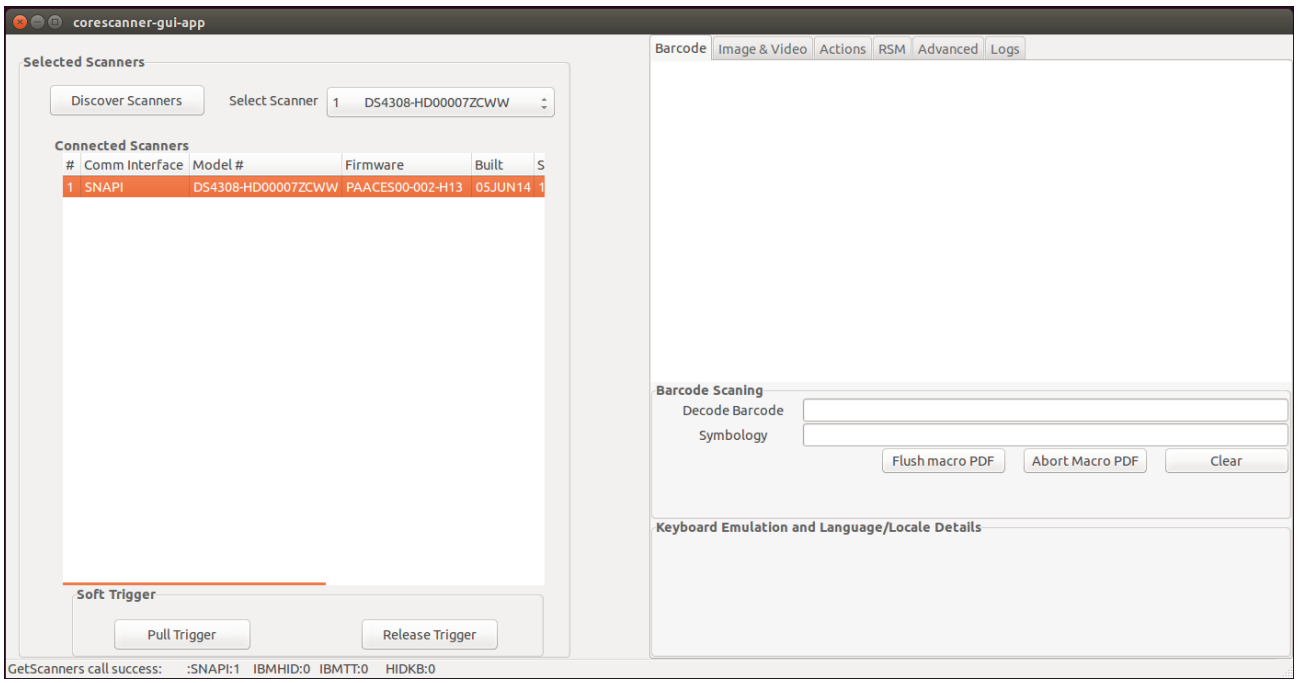


Figure 4-2 *Start Scanner SDK Sample Application (C++)*

4. Use the change directory (cd) command to change to the /usr/share/zebra-scanner/samples/gui-app directory and run the corescanner-gui-app executable file.

5. Click **Discover Scanners** to display all connected scanners. This executes an *Open* for all types of scanners and an *ExecCommand* with the *REGISTER_FOR_EVENTS* method using the following XML and a *GetScanners* API call:

```
<inArgs>
  <cmdArgs>
    <arg-int>6</arg-int>
    <arg-int>1,2,4,8,16,32</arg-int>
  </cmdArgs>
</inArgs>
```

NOTE The first <inArgs> tag in the XML contains the number of events to register, in this example, 6. The second <arg-int> tag contains the event IDs to register separated by commas (","). See *Table 4-2*.

Table 4-2 *Supported Event IDs*

Event Name	Event ID
SUBSCRIBE_BARCODE	1
SUBSCRIBE_IMAGE	2
SUBSCRIBE_VIDEO	4
SUBSCRIBE_RMD	8
SUBSCRIBE_PNP	16
SUBSCRIBE_OTHER	32

See *Chapter 5, SAMPLE SOURCE CODE* for more information about *Open*, *ExecCommand*, and *GetScanners* APIs.

The *GetScanners* API call produces the following XML code:

```
<?xml version="1.0" encoding="UTF-8" ?>
<scanners>
  <scanner type="SNAPI">
    <scannerID>1</scannerID>
    <serialnumber>7116000501003</serialnumber>
    <GUID>A2E647DED2163545B18BCEBD0A2A133D</GUID>
    <VID>1504</VID>
    <PID>6400</PID>
    <modelnumber>DS670-SR20001ZZR</modelnumber>
    <DoM>27APR07</DoM>
    <firmware>NBRPUAAC</firmware>
  </scanner>
</scanners>
```

Table 4-3 *GetScanners Output For This Example*

Scanner Information	Value	Description
Scanner ID	1	A unique ID that the SDK assigns to a scanner. Any scanner specific method executed via *ExecCommand* must point to this.
Serial Number	7116000501003	Device serial number printed on the label.
Model Number	DS670-SR20001ZZR	Device model number.
Date of Manufacture	27APR07	Device date of manufacture.
Firmware Version	NBRPUAAC	Current firmware version.
H/W GUID	A2E647DED2163545B18BCEBD0A2A133D	Hardware unique ID.

The XML consists of the scanner type, scanner ID, serial number, GUID, VID, PID, model number, date of manufacture, and firmware version of the connected scanners.

The *Connected Scanners* list displays all discovered scanners by processing the XML from the *GetScanners* command along with asset tracking information returned by querying device parameters. This information indicates the SDK was installed successfully.

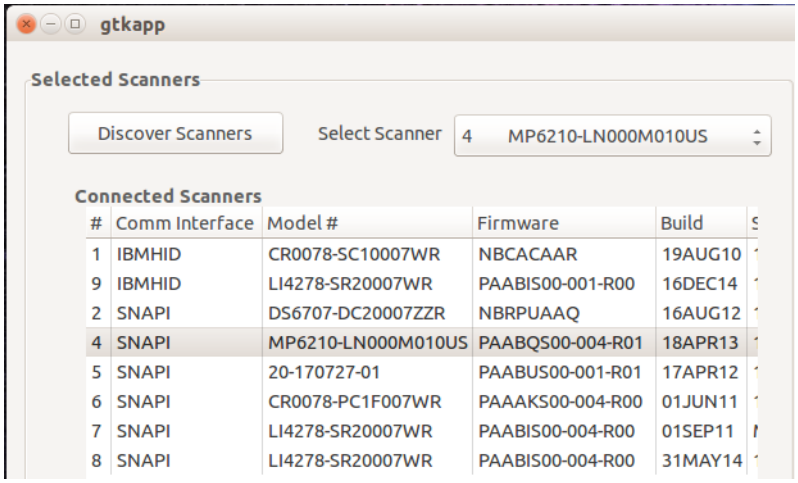


Figure 4-3 *Connected Scanners*

Barcode Scanning

Connect and discover a scanner. See *Scanner Discovery / Asset Tracking / Successful SDK Installation Validation on page 4-7*.

Scanning a barcode returns decoded data as XML on the *Barcode* tab. To illustrate the implementation, the sample application displays only the barcode data below the XML data.

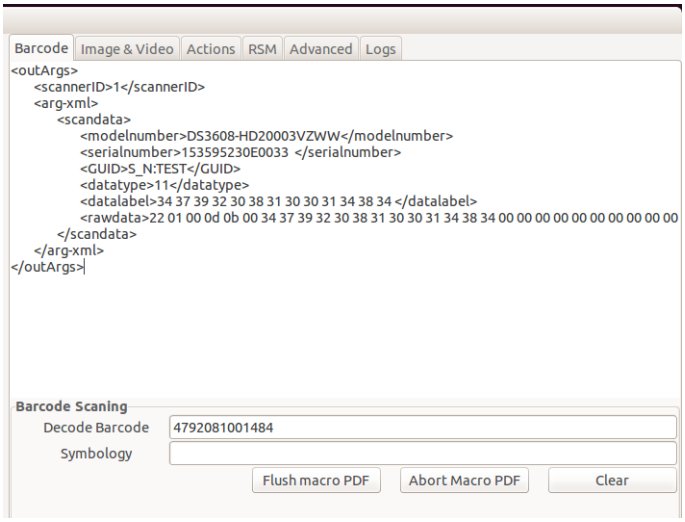


Figure 4-4 *Decoded Barcode Data*

Example

Scan the following barcode after discovering the scanner in the sample application.

Sample Barcode

This returns the following XML:

```
<?xml version="1.0" encoding="UTF-8" ?>
<outArgs>
  <scannerID>2</scannerID>
  <arg-xml>
    <scandata>
      <modelnumber>DS670-SR20001ZZR</modelnumber>
      <serialnumber>7116000501003</serialnumber>
      <GUID>A2E647DED2163545B18BCEBD0A2A133D</GUID>
      <datatype>8</datatype>
      <datalabel>0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x31 0x32</datalabel>
      <rawdata>0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x31 0x32</rawdata>
    </scandata>
  </arg-xml>
</outArgs>
```

The sample application processes this XML and displays the decoded barcode in the *Decoded Barcode* text box, and the symbology in the *Symbology* text box.

Image and Video Capture

1. Connect and discover an imaging scanner. See *Scanner Discovery / Asset Tracking / Successful SDK Installation Validation on page 4-7*.
2. Select a SNAPI mode scanner ID from the *Select Scanner* drop-down list. The selection appears in the *Connected Scanners* list.

> ***NOTE*** If no SNAPI scanner appears in the *Connected Scanners* list, connect an imaging scanner that supports image/video transfer. For a list if scanner models and their supported communication modes, refer to the Scanner SDK for Windows website: www.zebra.com/scannersdkforwindows.

Alternatively, select the SNAPI mode scanner in the *Connected Scanners* area. The selected scanner's ID appears in the *Select Scanner* drop-down list.

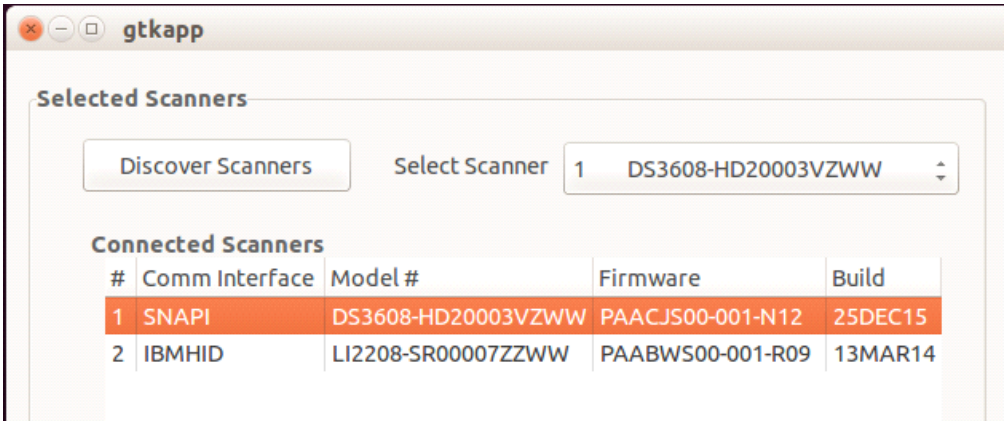


Figure 4-5 *Scanner Selection*

3. Select the *Image & Video* tab.
4. Select an image type of JPG, TIFF, or BMP. This executes an *ExecCommand* API call using the *ATTR_SET* method and following XML code:

```
<inArgs>
  <scannerID>1</scannerID>
  <cmdArgs>
    <arg-xml>
      <attrib_list>
        <attribute>
          <id>304</id>
          <datatype>B</datatype>
          <value>4</value>
        </attribute>
      </attrib_list>
    </arg-xml>
  </cmdArgs>
</inArgs>
```

> ***NOTE*** The <scannerID> tag in the XML contains the scanner ID selected in the *Connected Scanners* list. The <id> tag contains the image file type parameter of the selected scanner, in this example, 304. The value 4 indicates the image type the user gets from the scanner. See *Table 4-4*.

Table 4-4 *Image Types*

Image Type	Value
BMP_FILE_SELECTION	3
TIFF_FILE_SELECTION	4
JPEG_FILE_SELECTION	1

NOTE These values may vary by scanner model. Refer to the scanner Product Reference Guide for information on scanner parameters. For more information about parameter settings, see *Parameter Setting (Device Configuration) on page 4-20*.

5. Check *Enable Video View Finder*. This executes an *ExecCommand* API call with the *ATTR_SET* method and the following XML code:

```
<inArgs>
  <scannerID>1</scannerID>
  <cmdArgs>
    <arg-xml>
      <attrib_list>
        <attribute>
          <id>324</id>
          <datatype>B</datatype>
          <value>1</value>
        </attribute>
      </attrib_list>
    </arg-xml>
  </cmdArgs>
</inArgs>
```

NOTE The <scannerID> tag in the XML contains the ID of the scanner selected in the *Connected Scanners* list. The <id> tag contains the video view finder parameter number of the scanner and value 1 indicates that the view finder is enabled. A value 0 indicates the view finder is disabled.

6. Click either **Image** to put the scanner into image capture mode or **Video** to put the scanner into video capture mode. This executes an *ExecCommand* API call using the *DEVICE_CAPTURE_IMAGE* method for **Image**, or *DEVICE_CAPTURE_VIDEO* method for **Video**, with the following XML code:

```
<inArgs>
  <scannerID>1</scannerID>
</inArgs>
```

7. Click **Pull Trigger** on the bottom left side of the utility. The scanner captures an image if in image capture mode, or begins video capture if in video capture mode. Click **Release Trigger** to stop video capture.

   Clicking **Pull Trigger** or **Release Trigger** executes an *ExecCommand* API call using the corresponding *DEVICE_PULL_TRIGGER* or *DEVICE_RELEASE_TRIGGER* method with the following XML code:

```
<inArgs>
  <scannerID>1</scannerID>
</inArgs>
```

NOTE You can use the trigger on the scanner to start and stop image or video capture instead of the soft trigger buttons provided in the sample utility.

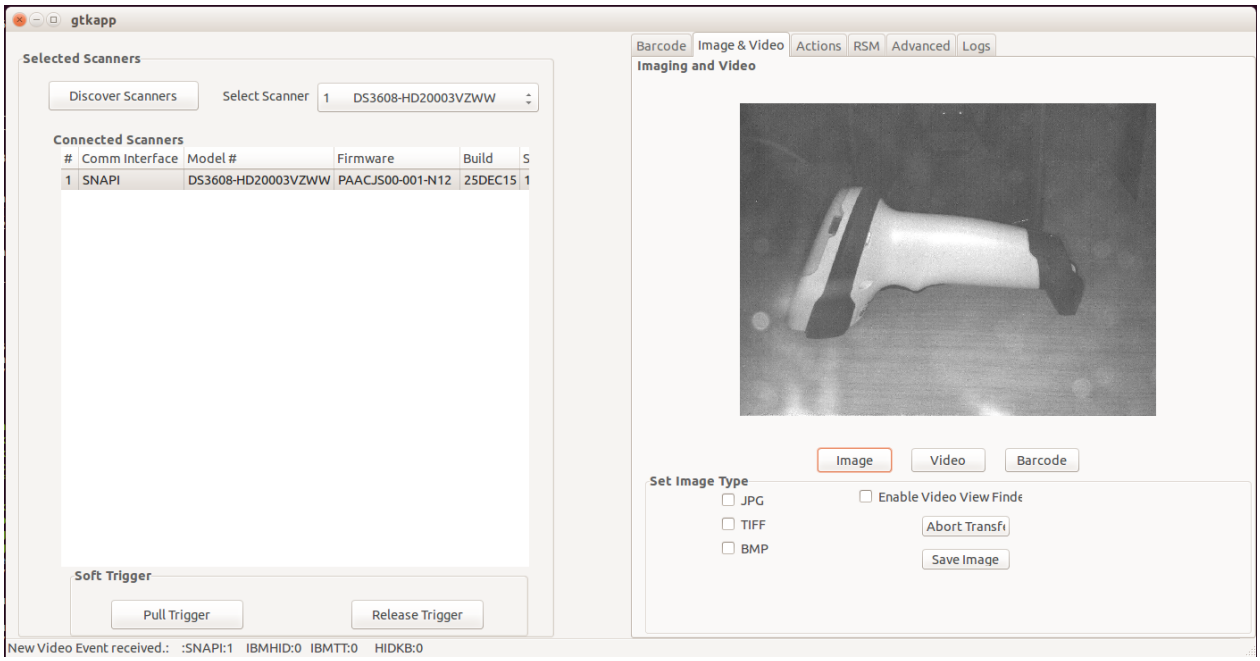


Figure 4-6 *Captured Image Displayed on the Image & Video Tab*

If you registered with ImageEvent you receive an image event for the pull trigger when in image mode. If you registered with VideoEvent you receive a video event for the pull trigger when in video mode. See *REGISTER_FOR_EVENTS on page 3-20*.

Beep the Beeper

Zebra scanners sound the beeper when the host system invokes the Beeper method.

1. Connect and discover a scanner. See *Scanner Discovery / Asset Tracking / Successful SDK Installation Validation on page 4-7*.
2. Select a SNAPI or IBM mode scanner ID from the *Select Scanner* drop-down list. The selection appears in the *Connected Scanners* list. See *Figure 4-5 on page 4-11*.
3. On the *Actions* tab, select the beep sequence.

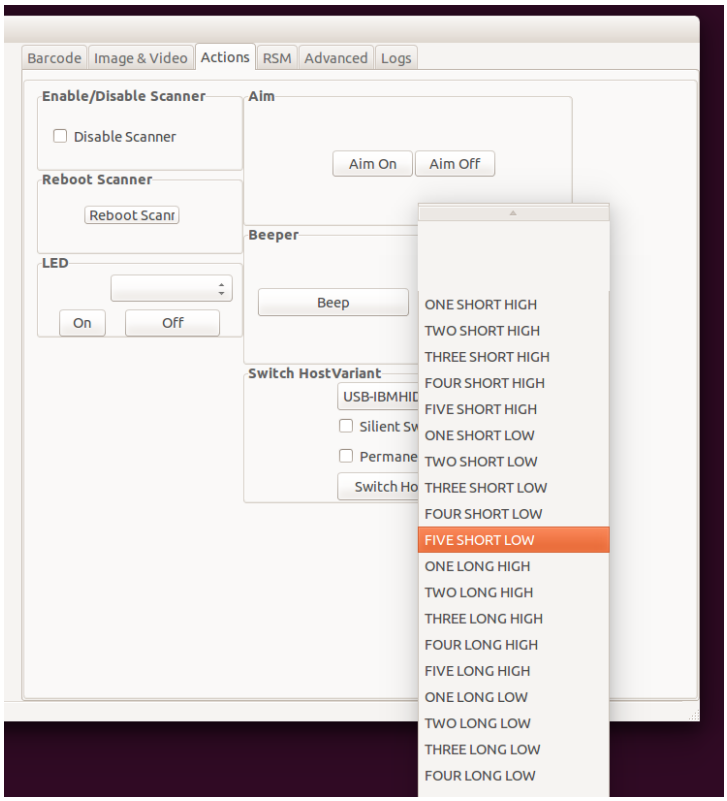


Figure 4-7 *Beep Values*

4. Click **Beep** to execute an *ExecCommand* API call with the *SET_ACTION* method and following XML code:

```
<inArgs>
  <scannerID>1</scannerID>
  <cmdArgs>
    <arg-int>2</arg-int>
  </cmdArgs>
</inArgs>
```

NOTE The <scannerID> tag in the XML contains the scanner ID selected in the *Connected Scanners* list. The <arg-int> tag in the XML contains the beep value selected in the *Beeper* drop-down list in *Figure 4-7*.

5. To sound a beep listed in *Table 3-12 on page 3-18*, change the value of the <arg-int> tag in the XML code. A successful command returns the status parameter 0.

Flash the LED

Zebra scanners flash an LED when the host system initiates the flash LED method.

1. Connect and discover a scanner. See *Scanner Discovery / Asset Tracking / Successful SDK Installation Validation on page 4-7*.
2. Select a SNAPI or IBM mode scanner ID from the *Select Scanner* drop-down list. The selection appears in the *Connected Scanners* list. See *Figure 4-5 on page 4-11*.
3. On the *Actions* tab, select the desired LED.

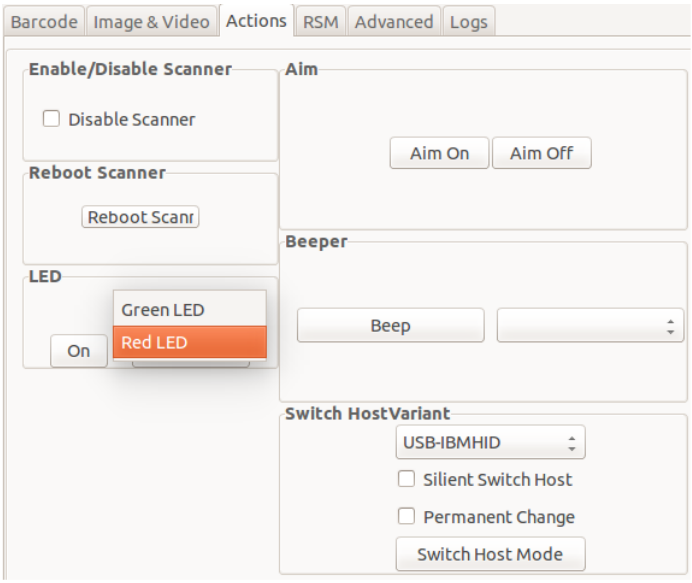


Figure 4-8 *LED Selection*

4. Click **On** to execute an *ExecCommand* API call with the *SET_ACTION* method and the following XML code:

```
<inArgs>
  <scannerID>1</scannerID>
  <cmdArgs>
    <arg-int>43</arg-int>
  </cmdArgs>
</inArgs>
```

NOTE The <scannerID> tag in the XML contains the scanner ID selected in the *Connected Scanners* list. The <arg-int> tag in the XML contains the corresponding action value to turn on or off the LED selected from the drop-down list in *Figure 4-8*.

To control the LED, change the action value in the <arg-int> tag. *Table 3-12 on page 3-18* lists the action values.

Click **Off** to execute an *ExecCommand* API call using the DEVICE_LED_OFF method with the same XML code that turned it on.

NOTE The *Beep the Beeper* and *Flash the LED* XML code examples are the same except for the method name. All XML used in an *ExecCommand* API call has a common format. The </inArgs> tag always contains the <scannerID> tag and optionally contains <cmdArgs> tags and <arg-xml> tags inside the </inArgs> tag. <cmdArgs> can contain <arg-string>, <arg-bool>, and <arg-int> tags. Execute different commands for the same XML by changing the method parameter in *ExecCommand*.

Attribute and Parameter Query

To query parameters from a specific device, such as the *Date of Manufacture* and *Firmware Version*:

1. Connect and discover a scanner. See *Scanner Discovery / Asset Tracking / Successful SDK Installation Validation on page 4-7*.
2. Select the scanner to query from the list of *Connected Scanners*.
3. On the *RSM* tab, click **Get All IDs** to retrieve a list of all attribute IDs supported by the scanner. This executes an *ExecCommand* API call with the *ATTR_GETALL* method and the following XML:

```
<inArgs><scannerID>1</scannerID></inArgs>
```

> ***NOTE*** The <scannerID> tag in the XML contains the ID of the scanner selected in the *Connected Scanners* list.

The sample application receives the following XML output and displays the corresponding attribute IDs. See *Figure 4-9 on page 4-18*.

```
<?xml version="1.0" encoding="UTF-8" ?>
<outArgs>
  <scannerID>1</scannerID>
  <arg-xml>
    <modelnumber>DS670-SR20001ZZR</modelnumber>
    <serialnumber>7116000501003</serialnumber>
    <GUID>A2E647DED2163545B18BCEBD0A2A133D</GUID>
    <response>
      <opcode>5000</opcode>
      <attrib_list>
        <attribute name="">0</attribute>
        <attribute name="">1</attribute>
        <attribute name="">2</attribute>
        <attribute name="">3</attribute>
        <attribute name="">4</attribute>
        <attribute name="">5</attribute>
        <attribute name="">6</attribute>
        <attribute name="">7</attribute>
        <attribute name="">8</attribute>
        <attribute name="">9</attribute>
        <attribute name="">10</attribute>
        <attribute name="">11</attribute>
        <attribute name="">12</attribute>
        <attribute name="">13</attribute>
        <attribute name="">14</attribute>
        <attribute name="">15</attribute>
        <attribute name="">16</attribute>
        <attribute name="">17</attribute>
        <attribute name="">18</attribute>
        <attribute name="">19</attribute>
        <attribute name="">20</attribute>
        <attribute name="">21</attribute>
        <attribute name="">22</attribute>
        <attribute name="">23</attribute>
        <attribute name="">24</attribute>
        <attribute name="">25</attribute>
        <attribute name="">26</attribute>
        <attribute name="">27</attribute>
        <attribute name="">28</attribute>
        <attribute name="">29</attribute>
```

```
            <attribute name="">30</attribute>
            <attribute name="">31</attribute>
            <attribute name="">34</attribute>
            <attribute name="">35</attribute>
            <attribute name="">36</attribute>
            <attribute name="">37</attribute>
            <attribute name="">38</attribute>
            <attribute name="">39</attribute>
            <attribute name="">655</attribute>
            <attribute name="">656</attribute>
            <attribute name="">657</attribute>
            <attribute name="">658</attribute>
            <attribute name="">659</attribute>
            <attribute name="">665</attribute>
            <attribute name="">670</attribute>
            <attribute name="">672</attribute>
            <attribute name="">673</attribute>
            <attribute name="">705</attribute>
            <attribute name="">716</attribute>
            <attribute name="">718</attribute>
            <attribute name="">721</attribute>
            <attribute name="">724</attribute>
            <attribute name="">726</attribute>
            <attribute name="">727</attribute>
            <attribute name="">728</attribute>
            <attribute name="">730</attribute>
            <attribute name="">731</attribute>
            <attribute name="">734</attribute>
            <attribute name="">735</attribute>
            <attribute name="">745</attribute>
            <attribute name="">6000</attribute>
            <attribute name="">6001</attribute>
            <attribute name="">6002</attribute>
            <attribute name="">6003</attribute>
            <attribute name="">6004</attribute>
            <attribute name="">20004</attribute>
            <attribute name="">20006</attribute>
            <attribute name="">20007</attribute>
            <attribute name="">20008</attribute>
            <attribute name="">20009</attribute>
            <attribute name="">20010</attribute>
            <attribute name="">20011</attribute>
            <attribute name="">20013</attribute>
          </attrib_list>
        </response>
      </arg-xml>
</outArgs>
```

NOTE To find the corresponding attribute names, refer to the scanner Product Reference Guide, available on the Zebra Support website. Attributes include configuration parameters, monitored data, and asset tracking information.

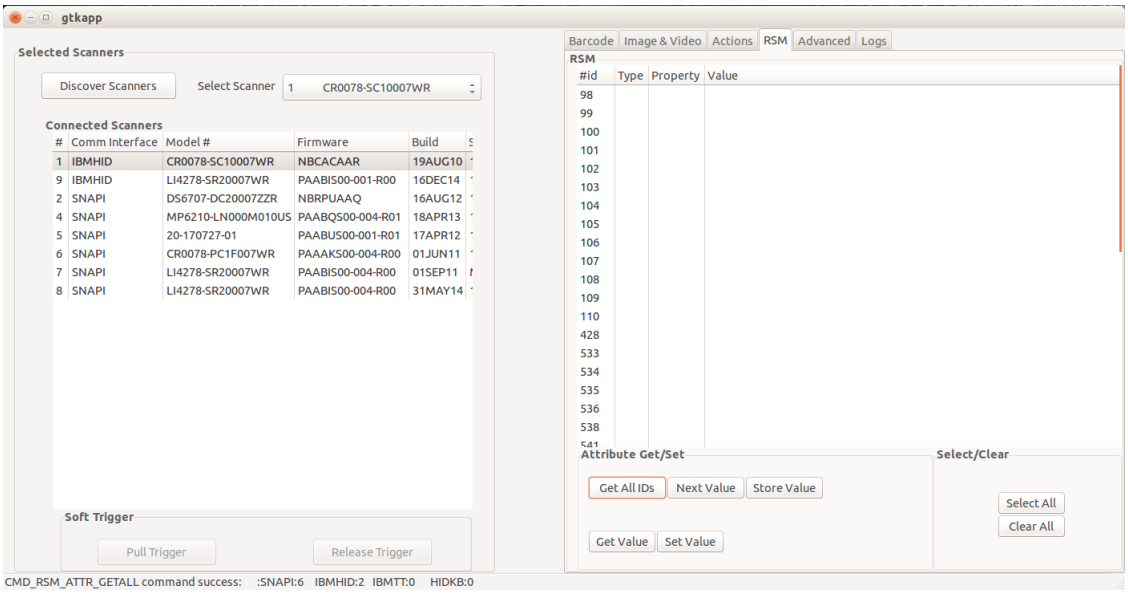


Figure 4-9 *Get RSM IDs*

Querying Specific Attributes

To query attributes, select attribute IDs and click **Get Value**. This executes an *ExecCommand* API call with the *ATTR_GET* method and the following XML.

```
<inArgs>
  <cmdArgs>
    <scannerID>1</scannerID>
    <arg-xml>
      <attrib_list>535,20004,1,140,392</attrib_list>
    </arg-xml>
  </cmdArgs>
</inArgs>
```

NOTE The <scannerID> tag in the XML contains the ID of the scanner selected in the *Connected Scanners* list and the <attrib_list> tag with the attribute IDs selected on the RSM tab.

For example, to retrieve the values of the *Date of Manufacture*, *Firmware Version*, *UPC -A status*, *Beeper Volume*, and *ADF Rule* parameters, select these attribute IDs and click **Get Value**. This executes an *ExecCommand* API call with the *ATTR_GET* method and the XML shown above. See *Table 4-5* for the corresponding IDs.

Table 4-5 *Device Parameters to Query*

Parameter	Attribute #
Date of Manufacture	535
Firmware Version	20004
UPC A status	1
Beeper Volume	140
ADF Rule	392

After a successful command, the output XML appears in the *Logs* tab as follows:


```
<?xml version="1.0" encoding="UTF-8" ?>
<outArgs>
  <scannerID>1</scannerID>
  <arg-xml>
    <modelnumber>DS670-SR20001ZZR</modelnumber>
    <serialnumber>7116000501003</serialnumber>
    <GUID>A2E647DED2163545B18BCEBD0A2A133D</GUID>
    <response>
      <opcode>5001</opcode>
      <attrib_list>
        <attribute>
          <id>535</id>
          <name></name>
          <datatype>S</datatype>
          <permission>R</permission>
          <value>27APR07</value>
        </attribute>
        <attribute>
          <id>20004</id>
          <name></name>
          <datatype>S</datatype>
          <permission>R</permission>
          <value>DS6707X4</value>
        </attribute>
        <attribute>
          <id>1</id>
          <name></name>
          <datatype>F</datatype>
          <permission>RWP</permission>
          <value>True</value>
        </attribute>
        <attribute>
          <id>140</id>
          <name></name>
          <datatype>B</datatype>
          <permission>RWP</permission>
          <value>0</value>
        </attribute>
        <attribute>
          <id>392</id>
          <name></name>
          <datatype>A</datatype>
          <permission>RWP</permission>
         <value>0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00</value>
        </attribute>
      </attrib_list>
    </response>
  </arg-xml>
</outArgs>
```

The sample application processes this XML and displays the output on the RSM tab.

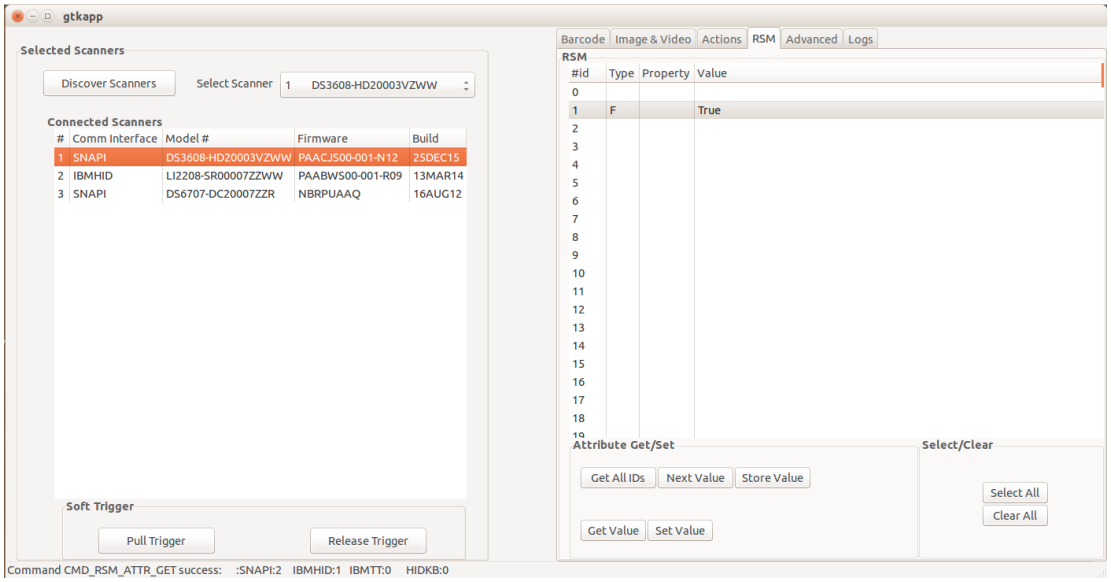


Figure 4-10 *Get Attribute Values*

Parameter Setting (Device Configuration)

To set parameters of a specific device, such as UPC-A status or beeper volume:

1. Query the parameter. See *Attribute and Parameter Query on page 4-16*.
2. To set an attribute, select and edit the attribute value in the *RSM* window.
3. Select the row of the updated attribute and click **Set Value** or **Store Value**. This executes an *ExecCommand* API call using the *ATTR_SET* or *ATTR_STORE* method and the following XML code:

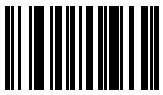
```
<inArgs>
  <scannerID>1</scannerID>
  <cmdArgs>
    <arg-xml>
      <attrib_list>
        <attribute>
          <id>1</id>
          <datatype>F</datatype>
          <value>False</value>
        </attribute>
      </attrib_list>
    </arg-xml>
  </cmdArgs>
</inArgs>
```

NOTE The <scannerID> tag in the XML contains the ID of the scanner selected in the *Connected Scanners* list and the <attrib_list> tag contains the <attribute> tags selected on the RSM tab.

Examples

The following examples demonstrate how to enable or disable a symbology, program an ADF rule, control beeper volume, and control LEDs.

Before starting an example, scan the **Set All Defaults** barcode to return all parameters to default values (replacing the scanner's current settings). Refer to the scanner Product Reference Guide for default values.



Set All Defaults

Enable / Disable a Symbology

To disable a symbology, refer to the scanner Product Reference Guide to determine the attribute ID. The attribute ID of the UPC-A parameter is 1. To change and validate the setting:

1. Put the scanner into USB OPOS (Hand-held) or USB SNAPI mode by scanning one of the barcodes in *Scanner Discovery / Asset Tracking / Successful SDK Installation Validation on page 4-7*, or using the procedure in *Host Variant Switching on page 4-25*.
2. Get the value of attribute ID 1, which is TRUE if you scanned the **Set All Defaults** barcode.
3. To disable UPC-A, change the value of attribute ID 1 to FALSE on the *RSM* tab, and click **Set Value** or **Store Value**. This executes an *ExecCommand* API call with the *ATTR_SET* or *ATTR_STORE* method and the XML shown in *Parameter Setting (Device Configuration) on page 4-20*.

If the command was successful, the scanner cannot scan the following UPC-A barcode.

Sample UPC-A Barcode

Program an ADF Rule

To create an ADF rule to add the prefix "A" to any barcode and an **Enter** key after, modify the scanner ADF buffer. The attribute ID of the ADF rule is 392.

To change and validate the setting:

USB Host Type = HID Keyboard Wedge

1. Scan the following barcode, or see *Host Variant Switching on page 4-25*, to switch the scanner to HID keyboard mode. This enables the scanner to send data to any text editor.

USB Host Type - HID Keyboard Wedge

2. Open a text editor such as Windows Notepad.
3. Scan the *Sample UPC-A Barcode on page 4-21* with the text editor as the active window to insert the barcode data "012345678912" in the window.
4. Put the scanner into USB OPOS (Hand-held) or USB SNAPI mode by scanning one of the barcodes in *Scanner Discovery / Asset Tracking / Successful SDK Installation Validation on page 4-7*, or using the procedure in *Host Variant Switching on page 4-25*.
5. In the sample application, change the value of the selected scanner's attribute 392 to:

   0x01 0x0C 0x11 0xF4 0x14 0x10 0x47 0x0D.

6. Click **Store Value**. The sample application executes an *ExecCommand* API call using the *ATTR_STORE* method and the following XML code:

```
<inArgs>
  <scannerID>1</scannerID>
  <cmdArgs>
    <arg-xml>
      <attrib_list>
        <attribute>
          <id>392</id>
          <datatype>A</datatype>
          <value>0x01 0x0C 0x11 0xF4 0x14 0x10 0x47 0x0D</value>
        </attribute>
      </attrib_list>
    </arg-xml>
  </cmdArgs>
</inArgs>
```

7. After successfully executing the command, repeat steps 1 - 3.

The text entered in Notepad is "A012345678912<Enter key>".

Beeper Volume Control

The beeper volume attribute ID is 140, and the scanner beeper has three volume levels:

- 2 = low
- 1 = Medium
- 0 = High

To change and validate this setting:

1. Put the scanner into USB OPOS (Hand-held) or USB SNAPI mode by scanning one of the barcodes in *Scanner Discovery / Asset Tracking / Successful SDK Installation Validation on page 4-7*, or using the procedure in *Host Variant Switching on page 4-25*.
2. Scan the *Sample UPC-A Barcode on page 4-21* and listen to the beeper.
3. Select attribute ID 140 from the RSM attribute grid. Its value is 0 if you scanned the **Set All Defaults** barcode at the beginning of the example.
4. Change the value to 2 and click **Set Value** or **Store Value**. The sample application executes an *ExecCommand* API call with the *ATTR_SET* or *ATTR_STORE* method and the following XML code:

```
<inArgs>
  <scannerID>1</scannerID>
  <cmdArgs>
    <arg-xml>
      <attrib_list>
        <attribute>
          <id>140</id>
          <datatype>B</datatype>
          <value>2</value>
        </attribute>
      </attrib_list>
    </arg-xml>
  </cmdArgs>
</inArgs>
```

5. Scan the *Sample UPC-A Barcode* again. The beeper volume is lower if the command was successful.

NOTE Changes made using the *Store Value* commands are permanent, persisting over power down and power up cycles. Changes made using the *Set Value* command are temporary and are discarded after the next power down.

Beeper and LED Control

Table 3-12 on page 3-18 indicates that the attribute ID for beeper and LED control is 6000. To change and validate this setting:

1. Put the scanner into USB OPOS (Hand-held) or USB SNAPI mode by scanning one of the barcodes in *Scanner Discovery / Asset Tracking / Successful SDK Installation Validation on page 4-7*, or using the procedure in *Host Variant Switching on page 4-25*.
2. To turn on the scanner LED, execute an *ExecCommand* API call with the *ATTR_SET* or *ATTR_STORE* method and the following XML code:

```
<inArgs>
  <scannerID>1</scannerID>
  <cmdArgs>
    <arg-xml>
      <attrib_list>
        <attribute>
          <id>6000</id>
          <datatype>X</datatype>
          <value>43</value>
        </attribute>
      </attrib_list>
    </arg-xml>
  </cmdArgs>
</inArgs>
```

NOTE You can execute several actions by changing the <value> tag in this XML. For example, to turn off the LED, change the value to 42; to beep the beeper, change the value to an integer between 0 and 25.

Host Variant Switching

1. Connect and discover a scanner. See *Scanner Discovery / Asset Tracking / Successful SDK Installation Validation on page 4-7*.
2. On the *Actions* tab, select a target mode from the *Switch Host Variant* drop-down list.
3. If desired, select *Permanent Change* or *Silent Reboot* options (hidden by the *Switch Host Variant* drop-down list).

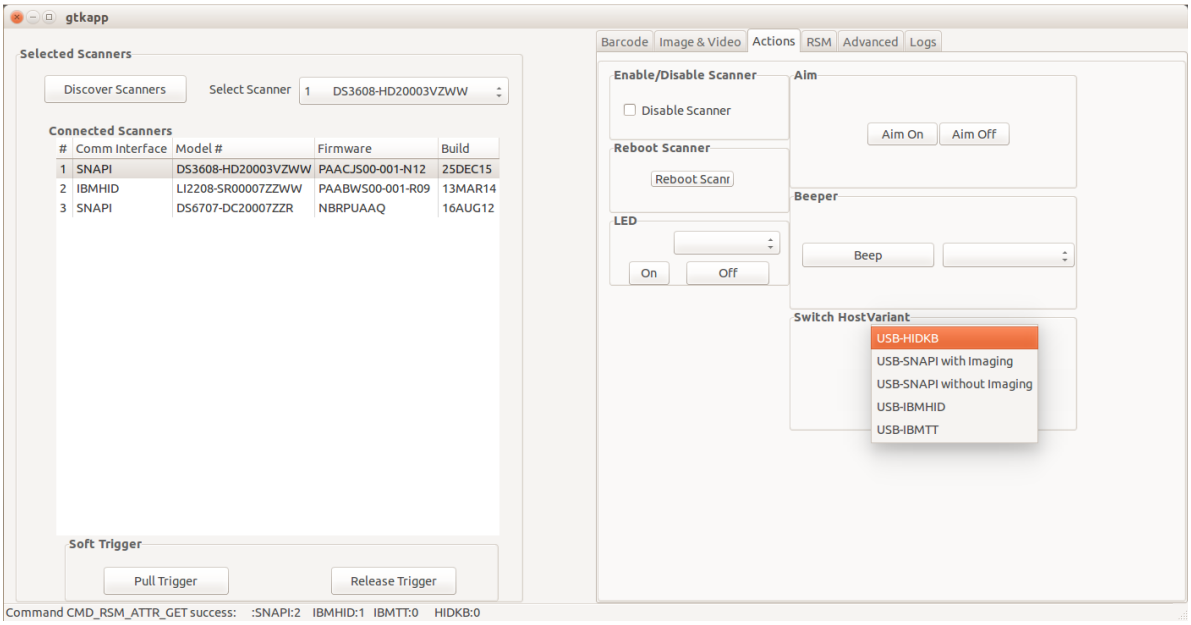


Figure 4-11 *Changing Host Mode*

4. Click **Switch Host Mode** to reboot the scanner and set the target mode. This executes an *ExecCommand* API call with the *DEVICE_SWITCH_HOST_MODE* method and the following XML code:

```
<inArgs>
  <scannerID>1</scannerID>
  <cmdArgs>
    <arg-string>XUA-45001-1</arg-string>
    <arg-bool>TRUE</arg-bool>
    <arg-bool>FALSE</arg-bool>
  </cmdArgs>
</inArgs>
```

NOTE The <scannerID> tag in the XML code contains the ID of the scanner selected in the *Connected Scanners* list. The <arg-string> tag contains the string code of the selected target host variant. See *Table 3-14 on page 3-36* for these codes.

The first <arg-bool> tag contains the boolean value for the silent reboot option. A value of TRUE silently reboots the scanner (without reboot beeps). The second <arg-bool> tag contains the boolean value for the permanent change option. A value of TRUE persists the target host variant over power down and power up cycles. Otherwise the host variant change is temporary until the next reboot.

HID Keyboard mode only supports the IBM Hand-held USB and SNAPI host variants. See *Table 3-14 on page 3-36* for USB host variant codes.

Firmware Upgrade

Firmware Upgrade Scenarios

There are three firmware upgrade scenarios.

Loading a compatible, different version of firmware to the scanner

Firmware upgrade downloads the firmware file to the scanner and activates the firmware. Activation takes approximately 50 seconds, during which the LED blinks red and the scanner does not respond to network queries.

When activation (programming) completes, the scanner reboots. The LED turns off, the scanner emits a power up beep, and powers up with the new firmware.

A firmware download can take up to 20 minutes depending on the connection speed between the POS terminal and the scanner, the operating mode of the scanner, and the size of the firmware file.

Loading the same version of firmware that is on the scanner

A firmware file can include multiple components. When loading the same version of firmware, some components in the firmware file may be the same as those on the scanner, while other components are different.

Before loading firmware, the scanner driver reads the header information of each firmware component to validate the model number and version, and only loads components that are different on the scanner. For example, if the first component downloading from the firmware file is the same version as one already on the scanner, the component does not load.

Loading an incompatible version of firmware on the scanner

This occurs when attempting to load firmware designed for one scanner model, for example a DS6707, onto another incompatible scanner model, for example the DS6708.

A firmware file can include multiple components. Before downloading firmware to the scanner, the scanner driver reads the header information of each firmware component to validate the model number and version. If the firmware component model number does not match the scanner, the component does not load. This process continues, verifying each remaining component in the firmware file.

Firmware Upgrade Procedure

1. Connect and discover a scanner. See*Scanner Discovery / Asset Tracking / Successful SDK Installation Validation on page 4-7*.
2. Use 123Scan^2 to obtain the latest scanner plug-in which contains the firmware .DAT file.
   a. Download and launch 123Scan^2 from www.zebra.com/123scan2.
   b. Using 123Scan^2, confirm you have the latest scanner plug-in, which contains a number of files including the firmware file and release notes.
   - To download the latest scanner plug-ins from within 123Scan^2, on the help menu, select **Check for updates**.
   - For a list of scanner models, plug-ins, and firmware files supported in 123Scan^2 on the help menu, select **Supported scanners and plug-ins**.
   - The plug-ins are located in one of the following 123Scan^2 sub folders:

     For Windows XP systems:
     [WINDOWSDRIVE]\ Documents and Settings\ [USERNAME]\Application Data\123Scan2\Plug-ins

     For later versions of Windows: *[WINDOWSDRIVE]\Users\ Application Data\123Scan2\Plug-ins*
3. Extract the firmware .DAT file from 123Scan^2 plug-in:
   a. Rename the file extension of the plug-in from .SCNPLG to .ZIP.
   b. Use a standard archive tool, such as WinZip, to extract the firmware update file which contains the file extension .DAT.

   For example, the DS9808 plug-in is named DS9808-COMMON SR MODELS-S-017.SCNPLG. Change .SCNPLG to .ZIP and access the firmware file CAAABS00-006-R02D0.DAT using WinZip.
4. In the sample application, select the *Advanced* tab and browse to and select the firmware .DAT file.
5. Check the *Bulk Update* option if desired.

   NOTE There are two communication interfaces (channels), USB HID or the faster USB Bulk, to perform a firmware update. Most SNAPI devices support USB Bulk firmware update but some do not. To determine if your scanner supports USB Bulk mode, refer to the Scanner SDK for Windows website: www.zebra.com/scannersdkforwindows.
6. Click **Update** to transfer the firmware file to the scanner. This executes an *ExecCommand* API call with the *UPDATE_FIRMWARE* method and the following XML code:

```
<inArgs>
  <scannerID>1</scannerID>
  <cmdArgs>
    <arg-string>D:\scanner\ScannerFW\DS6707\DS6707X4.DAT</arg-string>
    <arg-int>2</arg-int>
  </cmdArgs>
</inArgs>
```

   NOTE The <scannerID> tag in the XML contains the ID of the scanner selected in the *Connected Scanners* list. The <arg-string> tag contains the path to the firmware file. The <arg-int> tag contains 2 if the bulk update option is selected, or 1 if not.

7. If you have registered with *ScanRMDEvent* (see *REGISTER_FOR_EVENTS on page 3-20*), you receive six types of events per firmware update cycle, shown in *Table 4-6*.

Table 4-6 *Firmware Update Event Types*

Event Value	Event Type	Description
11	SCANNER_UF_SESS_START	Triggered when flash download session starts
12	SCANNER_UF_DL_START	Triggered when component download starts
13	SCANNER_UF_DL_PROGRESS	Triggered when block(s) of flash completed
14	SCANNER_UF_DL_END	Triggered when component download ends
15	SCANNER_UF_SESS_END	Triggered when flash download session ends
16	SCANNER_UF_STATUS	Triggered when update error or status

The OnScanRMDEvent function has two parameters:

- The first short type parameter contains the event type described above.
- The second parameter contains an XML for the event types in *Table 4-6*. Processing the XML provides further information. The receiving XML formats for each event types are as follows, and include information about the scanner it updates.

NOTE A firmware file can include multiple components. Before downloading firmware to the scanner, the scanner driver reads the header information of each firmware component to validate the model number and version. If the component model number does not match the scanner's, the component does not load. This process verifies each component in the firmware file.

a. SCANNER_UF_SESS_START

```
<?xml version="1.0" encoding="UTF-8"?>
<outArgs>
  <scannerID>1</scannerID>
  <arg-xml>
    <sess_start>
      <modelnumber>DS670-SR20001ZZR</modelnumber>
      <serialnumber>S/N:7108E15933CA1B4BB776F7BDB4B3F826</serialnumber>
      <GUID>7108E15933CA1B4BB776F7BDB4B3F826</GUID>
      <maxcount>3075</maxcount>
      <status>0</status>
    </sess_start>
  </arg-xml>
</outArgs>
```

NOTE The <maxcount> tag contains the number of records in the firmware file.

b. SCANNER_UF_DL_START

```
<?xml version="1.0" encoding="UTF-8"?>
<outArgs>
  <scannerID>1</scannerID>
  <arg-xml>
    <dl_start>
      <modelnumber>DS670-SR20001ZZR</modelnumber>
      <serialnumber>S/N:7108E15933CA1B4BB776F7BDB4B3F826</serialnumber>
      <GUID>7108E15933CA1B4BB776F7BDB4B3F826</GUID>
      <software_component>0</software_component>
      <status>0</status>
```

```
    </dl_start>
  </arg-xml>
</outArgs>
```

NOTE The <software_component> tag contains the component number that starts downloading.

c. SCANNER_UF_DL_PROGRESS

```
<?xml version="1.0" encoding="UTF-8"?>
<outArgs>
  <scannerID>1</scannerID>
  <arg-xml>
    <dl_progress>
      <modelnumber> DS670-SR20001ZZR </modelnumber>
      <serialnumber>S/N:7108E15933CA1B4BB776F7BDB4B3F826</serialnumber>
      <GUID>7108E15933CA1B4BB776F7BDB4B3F826</GUID>
      <software_component>1</software_component>
      <progress>7</progress>
      <status>600</status>
    </dl_progress>
    </arg-xml>
</outArgs>
```

NOTE The <progress> tag contains the record number that is currently downloading. The <status> tag contains the record of the download progress. 600 indicates the resident firmware. For status and error codes see *Table 3-15 on page 3-37*.

d. SCANNER_UF_DL_END

```
<?xml version="1.0" encoding="UTF-8"?>
<outArgs>
  <scannerID>1</scannerID>
  <arg-xml>
    <dl_end>
      <modelnumber> DS670-SR20001ZZR </modelnumber>
      <serialnumber>S/N:7108E15933CA1B4BB776F7BDB4B3F826</serialnumber>
      <GUID>7108E15933CA1B4BB776F7BDB4B3F826</GUID>
      <software_component>2</software_component>
      <size>0</size>
      <status>0</status>
    </dl_end>
    </arg-xml>
</outArgs>
```

e. SCANNER_UF_SESS_END

```
<?xml version="1.0" encoding="UTF-8"?>
<outArgs>
  <scannerID>1</scannerID>
  <arg-xml>
    <sess_end>
      <modelnumber> DS670-SR20001ZZR </modelnumber>
      <serialnumber>S/N:7108E15933CA1B4BB776F7BDB4B3F826</serialnumber>
      <GUID>7108E15933CA1B4BB776F7BDB4B3F826</GUID>
      <status>0</status>
    </sess_end>
    </arg-xml>
</outArgs>
```

f. SCANNER_UF_STATUS

```
<?xml version="1.0" encoding="UTF-8"?>
<outArgs>
  <scannerID>1</scannerID>
  <arg-xml>
    <sess_info>
      <modelnumber> DS670-SR20001ZZR </modelnumber>
      <serialnumber>S/N:7108E15933CA1B4BB776F7BDB4B3F826</serialnumber>
      <GUID>7108E15933CA1B4BB776F7BDB4B3F826</GUID>
      <status>506</status>
    </sess_info>
    </arg-xml>
</outArgs>
```

8. After file transfer completes, click **Launch** to activate (program into the scanner) the new firmware. Activation takes approximately one minute, during which the scanner blinks the red LED, cannot scan barcodes, and does not respond to network queries. When activation completes, the scanner reboots and turns off the LED, and then emits a power up beep and restarts with the new firmware.

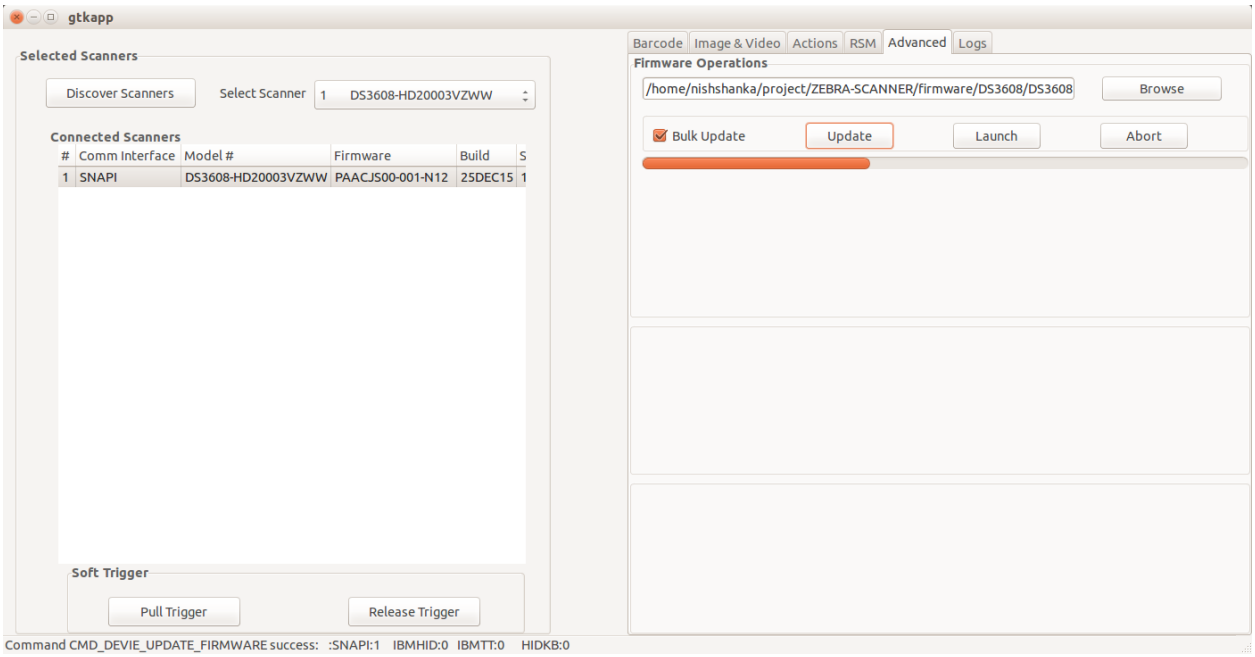


Figure 4-12 *Firmware Upgrade Through Bulk (Faster Download Mode) Channel*

CHAPTER 5 SAMPLE SOURCE CODE

Overview

This chapter describes how to use the Zebra Scanner SDK, and includes code snippets from the sample application.

NOTE For a list of a scanner's supported attribute (parameter) numbers and definitions, refer to the scanner Product Reference Guide, available on the Zebra Support website: http://www.zebra.com/support. Attributes include configuration parameters, monitored data, and asset tracking information.

SDK Sample Utilities

The Zebra Scanner SDK includes the following sample utilities that demonstrate SDK functionalities. This section describes how to use the utilities to gain an understanding of the SDK.

- Zebra Scanner SDK C++ GUI Demonstration Application, GTK (see *Scanner SDK C++ GTK-based Sample Application on page 4-4*)
- Zebra Scanner SDK C++ Console Sample Application, Commandline

Developing a Console Application Using zebra-scanner-devel and Libraries

zebra-scanner-sdk main packages include a development package that contains header files to access CoreScanner API calls, as well as source code for a sample application. The sample application can be modified and compiled using the make file to create a new application. Include a path for header files and LD_FLAGS.

NOTE The sample application demonstrates how to develop an application for Zebra scanner drivers.

Table 5-1 *Installation Locations for Sample Application, Development Resources, and Shared Libraries*

Application Type	Installation Location	File
Sample application	/usr/share/zebra-scanner/samples/console-app	Makefile
		ConsoleApplication
	/usr/share/zebra-scanner/samples/console-app/include	ConsoleSampleEventListener.h
		ConsoleMain.h
	/usr/share/zebra-scanner/samples/console-app/src	ConsoleSampleEventListener.cpp
		ConsoleMain.cpp
Development files for uPOS applications, files are installed by zebra-scanner-devel package	/usr/include/zebra-scanner	CsBarcodeTypes.h
		CsUserDefs.h
		CsIEventListenerXml.h
		Cslibcorescanner_xml.h
CoreScanner shared libraries	/usr/lib/zebra-scanner/corescanner	libcs-iudev.so.4.0.0
		libcs-comm.so.4.0.0
		libcs-common.so.4.0.0
		libcscl-snapi.so.4.0.0
		libcs-client.so.4.0.0
		libcs-clientscanner.so.4.0.0
		libcs-clientscale.so.4.0.0
		libcscl-ibmtt.so.4.0.0
		libcscl-ibmhh.so.4.0.0
		libcscl-hidkb.so.4.0.0
		libcs-client.so
		libcs-clientscale.so
		libcs-clientscanner.so
		libcs-comm.so
		libcs-common.so
		libcs-iudev.so
		libcscl-hidkb.so
		libcscl-snapi.so
		libcscl-ibmtt.so
		libcscl-ibmhh.so

An application should extend the IEventListenerXml class from the zebra-scanner-devel package. IEventListenerXml is an abstract class with pure virtual functions for all event handling. The included sample application implements all event handling methods. Additional methods support required features, and each method calls the ExecCommand() method after preparing input XML.

For example, ConsoleSampleEventListener.h is implemented in the IEventListenerXml class of the Console Application.

A developer implements all pure abstract methods inside the sample application's class SampleEventListener, which is derived from the IEventListenerXML interface. The SampleEventListener class can then be instantiated inside the main method. The following API calls are used for further development:

- Open
- GetScanners
- ExecCommand
- ExecCommandAsync
- Close

Source Code for the Console Application

Class SampleEventListener (ConsoleSampleEventListener.h)

```
/*
 * ©2015 Symbol Technologies LLC. All rights reserved.
 */

#ifndef SAMPLEEVENTLISTENER_H_
#define SAMPLEEVENTLISTENER_H_

#include "CsIEventListenerXml.h"
#include "CsUserDefs.h"
#include "CsBarcodeTypes.h"
#include "Cslibcorescanner_xml.h"

#include <iostream>

using namespace std;

class SampleEventListener : public IEventListenerXml
{
public:
    explicit SampleEventListener();
    virtual ~SampleEventListener();

    virtual void OnImageEvent( short eventType,
       int size, short imageFormat,
       char* sfimageData,
       int dataLength,
       std::string& pScannerData
       );
```

```
    virtual void OnVideoEvent( short eventType,
       int size, char* sfvideoData,
       int dataLength,
       std::string& pScannerData
       );

    virtual void OnBarcodeEvent( short eventType, std::string& pscanData );
    virtual void OnPNPEvent( short eventType, std::string ppnpData );
    virtual void OnCommandResponseEvent( short status, std::string& prspData );
    virtual void OnScannerNotification( short notificationType, std::string& pScannerData);
    virtual void OnIOEvent( short type, unsigned char data );
    virtual void OnScanRMDEvent( short eventType, std::string& prmdData );
    virtual void OnDisconnect();

    void Open();
    void GetScanners();
    void GetAttribute();
    void DiscoverTunnelingDevice();
    void GetAllAttributes();
    void SetAttribute();
    void SetAttributeStore();
    void SetZeroWeight();
    void Close();

    void RebootScanner();
    void ExecuteActionCommand(CmdOpcode opCode);
    void GetDeviceTopology();
    void FirmwareUpdate();
    void StartNewFirmware();
    void AbortFirmwareUpdate();
};


#endif /* SAMPLEEVENTLISTENER_H_ */
```

Class SampleEventListener Definitions/Implementation (ConsoleSampleEventListener.cpp)

```
/*
 * ©2015 Symbol Technologies LLC. All rights reserved.
 */

#include "ConsoleSampleEventListener.h"
#include <stdlib.h>
/**
 * Default constructor
 */
SampleEventListener::SampleEventListener()
{

}

/**
 * distructor
 */
SampleEventListener::~SampleEventListener()
```

```
{
   Close();
}

/**
 * Implementation of open()
 */
void SampleEventListener::Open()
{
    StatusID status;

    ::Open(this, SCANNER_TYPE_ALL, &status);

    std::string inXml =
"<inArgs><cmdArgs><arg-int>7</arg-int><arg-int>1,2,4,8,16,32,128</arg-int></cmdArgs>
</inArgs>";
    std::string outXml;
    ::ExecCommand(CMD_REGISTER_FOR_EVENTS, inXml, outXml, &status);
}

/**
 * implementation for GetScanners() method
 */
void SampleEventListener::GetScanners()
{
    unsigned short count;
    vector<unsigned int> list;
    string outXml;
    StatusID eStatus;

    ::GetScanners(&count, &list, outXml, &eStatus);

    cout << "GetScanners ******* Scanner Count: " << count << endl;
    cout << outXml << endl;
}

/**
 * Implementation of GetDeviceTopology()
 */
void SampleEventListener::GetDeviceTopology()
{
    string inXml;
    string outXml;
    StatusID sId;
    StatusID eStatus;

    cout << "GetDeviceTopology" << endl;
    ::ExecCommand(CMD_GET_DEVICE_TOPOLOGY, inXml, outXml, &sId);

    cout << "GetDeviceTopology" << endl;
    cout << outXml << endl;
}



/**
 * Implementation of GetAttribute()method
 */
void SampleEventListener::GetAttribute()
{
```

```
    //system("clear");
    cout << "================================" << endl;
    cout << "GetAttribute " << endl;
    std::string scannerID = "";
    cout << "Enter Scanner ID" << endl;
    cin >> scannerID;
    cout << "Enter attribute number or comma separated attribute numbers : " ;
    std::string attribute_number = "";
    cin >> attribute_number;

    std::string inXml = "<inArgs><scannerID>" + scannerID +
                        "</scannerID><cmdArgs><arg-xml><attrib_list>" +
                        attribute_number + "</attrib_list></arg-xml></cmdArgs></inArgs>";

    cout << "In XML : " << inXml << endl << endl;
    StatusID sId;
    std::string outXml;
    ::ExecCommand(CMD_RSM_ATTR_GET, inXml, outXml, &sId);

    cout << "Out XML : " << outXml << endl;
    cout << "================================" << endl;
}

/**
 * Implementation of GetAllAttributes()method
 */
void SampleEventListener::GetAllAttributes()
{
    std::string scannerID = "";
    std::string subscannerID = "";
    cout << "================================" << endl;
    cout << "Enter scanner ID: ";
    cin >> scannerID;

    std::string inXml = "<inArgs><scannerID>"+scannerID+"</scannerID></inArgs>";

    cout << "InXML : " << inXml << endl;
    StatusID sId;
    std::string outXml;

    ::ExecCommand(CMD_RSM_ATTR_GETALL, inXml, outXml, &sId);

    cout << "Out XML" << outXml << endl;
    cout << "================================" << endl;
}

void SampleEventListener::DiscoverTunnelingDevice()
{

    ::RefreshDevicelTopology();
}


/**
 * Implementation of SetAttribute()method
 */
void SampleEventListener::SetAttribute()
{
    //system("clear");
    cout << "================================" << endl;
```

```
    cout << "GetAttribute " << endl;
    std::string scannerID = "";
    cout << "Enter scanner ID : " << endl;
    cin >> scannerID;
    std::string attributeNumber = "";
    cout << "Enter attribute number : " << endl;
    cin >> attributeNumber;
    std::string dataType = "";
    cout << "Enter data type : " << endl;
    cin >> dataType;
    std::string attributeValue = "";
    cout << "Enter attribute value : " << endl;
    cin >> attributeValue;


    std::string inXml = "<inArgs><scannerID>"+ scannerID +
                        "</scannerID><cmdArgs><arg-xml><attrib_list><attribute><id>" +
attributeNumber +
                        "</id><datatype>" + dataType +
                        "</datatype><value>" + attributeValue +
                        "</value></attribute></attrib_list></arg-xml></cmdArgs></inArgs>";

    cout << "In XML : " << inXml << endl << endl;

    StatusID sId;

    std::string outXml;
    ::ExecCommand(CMD_RSM_ATTR_SET, inXml, outXml, &sId);

    cout << "Out XML : " << outXml << endl;

    cout << "================================" << endl;

}

/**
 * Implementation of SetAttributeStore() method
 */
void SampleEventListener::SetAttributeStore()
{
    cout << "================================" << endl;
    cout << "GetAttribute " << endl;
    std::string scannerID = "";
    cout << "Enter scanner ID : " << endl;
    cin >> scannerID;
    std::string attributeNumber = "";
    cout << "Enter attribute number : " << endl;
    cin >> attributeNumber;
    std::string dataType = "";
    cout << "Enter data type : " << endl;
    cin >> dataType;
    std::string attributeValue = "";
    cout << "Enter attribute value : " << endl;
    cin >> attributeValue;


    std::string inXml = "<inArgs><scannerID>"+ scannerID +
                        "</scannerID><cmdArgs><arg-xml><attrib_list><attribute><id>" +
attributeNumber +
                        "</id><datatype>" + dataType +
```

```
                        "</datatype><value>" + attributeValue +
                        "</value></attribute></attrib_list></arg-xml></cmdArgs></inArgs>";

    cout << "In XML : " << inXml << endl << endl;

    StatusID sId;

    std::string outXml;
    ::ExecCommand(CMD_RSM_ATTR_STORE, inXml, outXml, &sId);

    cout << "Out XML : " << outXml << endl;

    cout << "================================" << endl;
}


void SampleEventListener::SetZeroWeight()
{
    std::string inXml =
"<inArgs><scannerID>1</scannerID><cmdArgs><arg-xml><attrib_list><attribute><id>6019</id><d
atatype>X</datatype><value>0</value></attribute></attrib_list></arg-xml></cmdArgs></inArgs
>";

    cout << "In XML : " << inXml << endl << endl;
    StatusID sId;
    std::string outXml;
    ::ExecCommand(CMD_RSM_ATTR_SET, inXml, outXml, &sId);

    cout << "Out XML : " << outXml << endl;

}


void SampleEventListener::Close()
{
    StatusID status;
    ::Close(0, &status);

}


void SampleEventListener::OnImageEvent( short eventType, int size, short imageFormat,
char* sfimageData, int dataLength, std::string& pScannerData )
{
   cout << "OnImageEvent" << endl;
}


void SampleEventListener::OnVideoEvent( short eventType, int size, char* sfvideoData, int
dataLength, std::string& pScannerData )
{
   cout << "OnVideoEvent" << endl;
}


void SampleEventListener::OnPNPEvent( short eventType, std::string ppnpData )
{
    string str;
    if (eventType == SCANNER_ATTACHED) {
        str = ppnpData;
```

```
    } else if (eventType == SCANNER_DETACHED) {
        str =  ppnpData;
    } else {
        str = " UNKNOWN PNP Event ";
    }
    cout << str << endl;
}


void SampleEventListener::OnCommandResponseEvent( short status, std::string& prspData )
{
    cout << endl << "Scanner data: " << prspData << endl;
   cout << "OnCommandResponseEvent" << endl;
   cout << prspData << endl;
}


void SampleEventListener::OnScannerNotification( short notificationType, std::string&
pScannerData )
{
    cout << endl << "Scanner event data: " << pScannerData << endl;
   cout << "OnScannerNotification" << endl;
}


void SampleEventListener::OnIOEvent( short type, unsigned char data )
{
   cout << "OnIOEvent" << endl;
}

void SampleEventListener::OnScanRMDEvent( short eventType, std::string& prmdData )
{
   cout << "OnScanRMDEvent" << endl;
        cout << "Out XML " << endl;
        cout << prmdData << endl;
}

void SampleEventListener::OnDisconnect()
{
   cout << "OnDisconnect" << endl;
}

void SampleEventListener::OnBarcodeEvent(short int eventType, std::string & pscanData)
{
   cout << "Barcode Detected" << endl;
   cout << "Out XML" << endl;
   cout << pscanData << endl;
}

void SampleEventListener::RebootScanner()
{

}

void SampleEventListener::FirmwareUpdate()
{
    std::string inXml;
    std::string outXml;
    std::string datFilePath;
    StatusID sId;
```

```
    StatusID eStatus;
    std::string scannerID;
    std::string bulkOption;

    cout << "================================" << endl;
    std::cout << "FirmwareUpdate" << std::endl;
    std::cout << "Enter Scanner ID: " << std::endl;
    std::cin >> scannerID;
    std::cout << "Enter Firmware DAT file path: " << std::endl;
    std::cin >> datFilePath;
    std::cout << "Enter USB communication mode 1=hid, 2=bulk : ";
    std::cin >> bulkOption;

    inXml = "<inArgs><scannerID>" + scannerID + "</scannerID><cmdArgs><arg-string>" +
datFilePath + "</arg-string><arg-int>" + bulkOption + "</arg-int></cmdArgs></inArgs>";

    cout << "InXML : " << inXml << endl;

    ::ExecCommand(CMD_DEVICE_UPDATE_FIRMWARE, inXml, outXml, &sId);

    cout << "================================" << endl;
    cout << outXml << endl;
}

void SampleEventListener::FirmwareUpdateFromPlugin()
{
     std::string inXml;
    std::string outXml;
    std::string pluginFilePath="";
    StatusID sId;
    StatusID eStatus;
    std::string scannerID;
    std::string bulkOption;

    cout << "================================" << endl;
    std::cout << "FirmwareUpdate From Plugin." << std::endl;
    std::cout << "Enter Scanner ID: " << std::endl;
    std::cin >> scannerID;
    std::cout << "Enter Firmware Pugin file path: " << std::endl;

    while ( pluginFilePath.size() < 4 ){
        std::getline(std::cin, pluginFilePath);

    }

    if ( !(pluginFilePath.substr(pluginFilePath.find_last_of(".")+ 1) == "SCNPLG") ){
        std::cout << "Please Enter a file with extension .SCNPLG." << std::endl;
        return;
    }

    std::cout << "Enter USB communication mode 1=hid, 2=bulk : ";
    std::cin >> bulkOption;

    inXml = "<inArgs><scannerID>" + scannerID + "</scannerID><cmdArgs><arg-string>" +
pluginFilePath + "</arg-string><arg-int>" + bulkOption + "</arg-int></cmdArgs></inArgs>";

    cout << "InXML : " << inXml << endl;

    ::ExecCommand(CMD_DEVICE_UPDATE_FIRMWARE_FROM_PLUGIN, inXml, outXml, &sId);
```

```
    cout << "================================" << endl;
    cout << outXml << endl;
}

void SampleEventListener::StartNewFirmware()
{
    std::string inXml;
    std::string outXml;
    StatusID sId;
    StatusID eStatus;
    std::string scannerID;

    cout << "================================" << endl;
    std::cout << "Starting new firmware" << std::endl;
    std::cout << "Enter Scanner ID: " << std::endl;
    std::cin >> scannerID;
    std::cout << "Enter Firmware DAT file path: " << std::endl;

    inXml = "<inArgs><scannerID>" + scannerID + "</scannerID></inArgs>";

    cout << "InXML : " << inXml << endl;

    ::ExecCommand(CMD_START_NEW_FIRMWARE, inXml, outXml, &sId);

    cout << "================================" << endl;
    cout << outXml << endl;
}

void SampleEventListener::AbortFirmwareUpdate()
{
    std::string inXml;
    std::string outXml;
    StatusID sId;
    StatusID eStatus;
    std::string scannerID;

    cout << "================================" << endl;
    std::cout << "Abort Firmware Update" << std::endl;
    std::cout << "Enter Scanner ID: " << std::endl;
    std::cin >> scannerID;

    inXml = "<inArgs><scannerID>" + scannerID + "</scannerID></inArgs>";

    cout << "InXML : " << inXml << endl;

    ::ExecCommand(CMD_DEVICE_ABORT_UPDATE_FIRMWARE, inXml, outXml, &sId);

    cout << "================================" << endl;
    cout << outXml << endl;
}

/**
 * Method to execute action attribute related commands.
 * added to v1.3.0 release.
 * @param opCode
 */
void SampleEventListener::ExecuteActionCommand(CmdOpcode opCode)
{
    std::string scannerID = "";
    std::string ledNumber = "";
```

```
    std::string beeperCode = "";
    std::string inXml;

    cout << "================================" << endl;

    switch (opCode)
    {
        case CMD_DEVICE_LED_ON:
        case CMD_DEVICE_LED_OFF:
        {
            cout << "Enter scanner ID: ";
            cin >> scannerID;
            cout << "Enter LED number: ";
            cin >> ledNumber;

            inXml = "<inArgs><scannerID>" + scannerID +
                                "</scannerID><cmdArgs><arg-int>" + ledNumber +
                                "</arg-int></cmdArgs></inArgs>";
            break;
        }
        case CMD_DEVICE_BEEP_CONTROL:
        {
            cout << "Enter scanner ID: ";
            cin >> scannerID;
            cout << "Enter Beeper code: ";
            cin >> beeperCode;

            inXml = "<inArgs><scannerID>" + scannerID +
                                "</scannerID><cmdArgs><arg-int>" + beeperCode +
                                "</arg-int></cmdArgs></inArgs>";
            break;
        }
        default:
        {
            cout << "Enter scanner ID: ";
            cin >> scannerID;

            inXml = "<inArgs><scannerID>" + scannerID + "</scannerID></inArgs>";
            break;
        }
    }

    cout << "InXML : " << inXml << endl;
    StatusID sId;
    std::string outXml;

    ::ExecCommand(opCode, inXml, outXml, &sId);

    cout << "Out XML" << outXml << endl;
    cout << "================================" << endl;
}
```

After implementation for the class IEventListenerXml, the main class can create an instance of the class SampleEventListener for the demonstration application. It is mandatory to implement IEventListenerXml class because it is an abstract class provided by zebra-scanner. Following is main.cpp for the sample application.

Main Method Implementation

```
/*
 * ©2015 Symbol Technologies LLC. All rights reserved.
 */

#include "ConsoleMain.h"
#include "ConsoleSampleEventListener.h"

#include <stdlib.h>
//class LoggingContext;

bool hasAlreadyOpen = false;

int ReturnChoice()
{
    cout << "======Select CoreScanner command====" << endl;
    cout << "=== 1. GetAttribute " << endl;
    cout << "=== 2. GetAllAttributes " << endl;
    cout << "=== 3. Get Scanners " << endl;
    cout << "=== 4. SetAttribute " << endl;
    cout << "=== 5. SetAttribute-Store " << endl;
    cout << "=== 6. Device LED ON or OFF " << endl;
    cout << "=== 7. Scanner Enable " << endl;
    cout << "=== 8. Scanner Disable " << endl;
    cout << "=== 9. Scanner Reboot " << endl;
    cout << "== 10. Refresh Device Topology" << endl;
    cout << "== 11. Action - Beeper command" << endl;
    cout << "== 12. AIM ON" << endl;
    cout << "== 13. AIM OFF" << endl;
    cout << "====================================" << endl;
    cout << "=== 0. Exit" << endl;
    cout << "= 100. Main Menu" << endl;
    cout << "====================================" << endl;
    cout << "Enter choice : " ;

    int choice = 0;
    cin >> choice;
    return choice;
}

int main(void)
{
     cout << "Zebra Scanner Sample Application" << endl;
     cout << "====================================" << endl;

    SampleEventListener sel;

    sel.Open();
    sel.GetScanners();

    int choice = ReturnChoice();

    do {
        switch (choice) {
            case 1:
                sel.GetAttribute();
                break;
            case 2:
                sel.GetAllAttributes();
```

```
                break;
            case 3:
                sel.GetScanners();
                break;
            case 4:
                sel.SetAttribute();
                break;
            case 5:
                sel.SetAttributeStore();
                break;
            case 6:
                sel.ExecuteActionCommand(CMD_DEVICE_LED_ON);
                break;
            case 7:
                sel.ExecuteActionCommand(CMD_DEVICE_SCAN_ENABLE);
                break;
            case 8:
                sel.ExecuteActionCommand(CMD_DEVICE_SCAN_DISABLE);
                break;
            case 9:
                sel.ExecuteActionCommand(CMD_REBOOT_SCANNER);
                break;
            case 10:
                sel.DiscoverTunnelingDevice();
                break;
            case 11:
                sel.ExecuteActionCommand(CMD_DEVICE_BEEP_CONTROL);
                break;
            case 12:
                sel.ExecuteActionCommand(CMD_DEVICE_AIM_ON);
                break;
            case 13:
                sel.ExecuteActionCommand(CMD_DEVICE_AIM_OFF);
                break;
            case 100:
                cout << "====================================" << endl;
                break;
            case 0:
            default:
                break;
        }

         if(choice != 0)
             choice = ReturnChoice();

} while (choice);
    sel.Close();
    return 0;
}
```

Calling Open Command

```
void SampleEventListener::Open()
{
    StatusID status;
    ::Open(this, SCANNER_TYPE_ALL, &status);

    // register for all events.
    std::string inXml = "<inArgs><cmdArgs><arg-int>7</arg-int><arg-int>1,2,4,8,16,32,128
   </arg-int></cmdArgs></inArgs>";

    std::string outXml;
    ::ExecCommand(CMD_REGISTER_FOR_EVENTS, inXml, outXml, &status);
}
```

Calling Close Command

```
void SampleEventListener::Close()
{
    StatusID status;
        ::Close(0, &status);
}
```

Calling GetScanners Command

```
void SampleEventListener::GetScanners()
{
    unsigned short count;
    vector<unsigned int> list;
    string outXml;
    StatusID eStatus;

    ::GetScanners(&count, &list, outXml, &eStatus);

    cout << "GetScanners" << endl;
    cout << outXml << endl;
}
```

Calling ExecCommand Command and ExecCommandAsync Command

```
void SampleEventListener::GetAttribute()
{
    cout << "================================" << endl;
    cout << "GetAttribute " << endl;
    std::string scannerID = "";
    cout << "Enter Scanner ID" << endl;
    cin >> scannerID;
    cout << "Enter attribute number or comma separated attribute numbers : " ;
    std::string attribute_number = "";
    cin >> attribute_number;

    std::string inXml = "<inArgs><scannerID>" + scannerID +
                        "</scannerID><cmdArgs><arg-xml><attrib_list>" +
                        attribute_number + "</attrib_list></arg-xml></cmdArgs></inArgs>";

    cout << "In XML : " << inXml << endl << endl;
    StatusID sId;
    std::string outXml;
    ::ExecCommand(CMD_RSM_ATTR_GET, inXml, outXml, &sId);

    cout << "Out XML : " << outXml << endl;
    cout << "================================" << endl;
}
```

Quick Startup

Zebra Technologies Corporation
Lincolnshire, IL U.S.A.
http://www.zebra.com

MN-003028-03 Revision A - May 2019