



MOTOROLA SCANNER SDK FOR WINDOWS DEVELOPER'S GUIDE

MOTOROLA SCANNER SDK FOR WINDOWS DEVELOPER'S GUIDE

72E-149784-01

Rev. A

June 2011

No part of this publication may be reproduced or used in any form, or by any electrical or mechanical means, without permission in writing from Motorola. This includes electronic or mechanical means, such as photocopying, recording, or information storage and retrieval systems. The material in this manual is subject to change without notice.

The software is provided strictly on an “as is” basis. All software, including firmware, furnished to the user is on a licensed basis. Motorola grants to the user a non-transferable and non-exclusive license to use each software or firmware program delivered hereunder (licensed program). Except as noted below, such license may not be assigned, sublicensed, or otherwise transferred by the user without prior written consent of Motorola. No right to copy a licensed program in whole or in part is granted, except as permitted under copyright law. The user shall not modify, merge, or incorporate any form or portion of a licensed program with other program material, create a derivative work from a licensed program, or use a licensed program in a network without written permission from Motorola. The user agrees to maintain Motorola's copyright notice on the licensed programs delivered hereunder, and to include the same on any authorized copies it makes, in whole or in part. The user agrees not to decompile, disassemble, decode, or reverse engineer any licensed program delivered to the user or any portion thereof.

Motorola reserves the right to make changes to any software or product to improve reliability, function, or design.

Motorola does not assume any product liability arising out of, or in connection with, the application or use of any product, circuit, or application described herein.

No license is granted, either expressly or by implication, estoppel, or otherwise under any Motorola, Inc., intellectual property rights. An implied license only exists for equipment, circuits, and subsystems contained in Motorola products.

Revision History

Changes to the original guide are listed below:

Change	Date	Description
-01 Rev A	6/2011	Initial release.

TABLE OF CONTENTS

Revision History	iii
------------------------	-----

About This Guide

Introduction	ix
Chapter Descriptions	ix
Notational Conventions	x
Service Information	xi

Chapter 1: INTRODUCTION TO THE MOTOROLA SCANNER SDK

Overview	1-1
Quick Startup	1-3
FAQs	1-4
Motorola Scanner SDK Architecture	1-5
Multiple Scanner Device Identification Methodology For Applications	1-7
How Multiple Applications Access Multiple Scanners From Scanner SDK	1-7
Three Applications Connected To One Scanner	1-7
Implementation Details	1-7
Three Applications Connected To Two Scanners	1-8
Implementation Details	1-8
Many-to-Many Application Device Usage	1-8
Implementation Details	1-8
One Application Connected to Two Scanners	1-9
Implementation Details	1-9

Chapter 2: INSTALLATION & CONFIGURATION

Overview	2-1
SDK Components	2-2
System Requirements	2-2
Supported Operating Systems	2-2
Scanner Models Versus Communication Modes	2-3
Installing the SDK	2-4

Step-by-Step Installation Instructions	2-4
Installed Components	2-9
Configuration	2-11
Simulated HID Keyboard Output	2-11
Sample config.xml	2-11
Basic Installation Verification	2-12

Chapter 3: MOTOROLA SCANNER SDK API

Overview	3-1
API Commands	3-2
Open	3-2
GetScanners	3-3
ExecCommand	3-3
ExecCommandAsync	3-4
Close	3-4
API Events	3-5
ImageEvent	3-5
Syntax	3-5
Parameters	3-5
VideoEvent	3-6
Syntax	3-6
Parameters	3-6
BarcodeEvent	3-7
Syntax	3-7
Parameters	3-7
PNPEvent	3-10
Syntax	3-10
Parameters	3-10
ScanRMDEvent	3-11
Syntax	3-11
Parameters	3-11
CommandResponseEvent	3-11
Syntax	3-11
Parameters	3-11
Methods Invoked Through ExecCommand Or ExecCommandAsync	3-12
Examples: Using the Methods	3-13
GET_VERSION	3-13
REGISTER_FOR_EVENTS	3-14
UNREGISTER_FOR_EVENTS	3-14
CLAIM_DEVICE	3-15
RELEASE_DEVICE	3-15
ABORT_MACROPDF	3-15
ABORT_UPDATE_FIRMWARE	3-15
AIM_OFF	3-16
AIM_ON	3-16
FLUSH_MACROPDF	3-16
DEVICE_LED_OFF	3-17
DEVICE_LED_ON	3-17
DEVICE_PULL_TRIGGER	3-17
DEVICE_RELEASE_TRIGGER	3-18

SCAN_DISABLE	3-18
SCAN_ENABLE	3-18
SET_PARAMETER_DEFAULTS	3-18
DEVICE_SET_PARAMETERS	3-19
SET_PARAMETER_PERSISTANCE	3-19
DEVICE_BEEP_CONTROL	3-20
REBOOT_SCANNER	3-21
DEVICE_CAPTURE_IMAGE	3-21
DEVICE_CAPTURE_VIDEO	3-22
ATTR_GETALL	3-22
ATTR_GET	3-24
ATTR_GETNEXT	3-25
ATTR_SET	3-26
ATTR_STORE	3-27
GET_DEVICE_TOPOLOGY	3-27
START_NEW_FIRMWARE	3-28
UPDATE_FIRMWARE	3-28
DEVICE_SET_SERIAL_PORT_SETTINGS	3-29
DEVICE_SWITCH_HOST_MODE	3-29
KEYBOARD_EMULATOR_ENABLE	3-30
KEYBOARD_EMULATOR_SET_LOCALE	3-30
KEYBOARD_EMULATOR_GET_CONFIG	3-31
Error/Status Codes	3-32

Chapter 4: TEST UTILITIES & SOURCE CODE

Overview	4-1
Test Utilities Provided in the SDK	4-2
Motorola Scanner SDK C++ Sample Application	4-2
Motorola Scanner SDK C#.Net Sample Application	4-3
How to Verify Scanner SDK Functionality	4-6
Scanner Discovery / Asset Tracking Information / Validating Successful SDK Installation	4-6
Bar Code Scanning	4-8
Example	4-9
Language/Locale Details	4-9
Capture Image and Video	4-10
Beep the Beeper	4-13
Flash the LED	4-14
Querying Attributes and Parameters	4-15
Parameter Setting (Device Configuration)	4-19
Examples	4-20
Enable / Disable a Symbology	4-20
Programming an ADF Rule	4-21
Beeper Volume Control	4-22
Beeper and LED Control	4-22
Host Variant Switching	4-23
Firmware Upgrade	4-24
Firmware Upgrade Scenarios	4-24
Firmware Upgrade Procedure	4-25

Chapter 5: SAMPLE SOURCE CODE

Overview	5-1
Sample Utilities Provided in the SDK	5-1
Creation of COM Object And Registration for Events	5-1
Register for COM Events	5-2
Calling Open Command	5-2
Calling Close Command	5-2
Calling GetScanners Command	5-3
Calling ExecCommand Command and ExecCommandAsync Command	5-3

Appendix A: WRITE SIMPLE APPLICATIONS USING CORE SCANNER APIS

Overview	A-1
Import CoreScanner Reference, Class Declaration and Instantiation	A-2
Call Open API	A-6
Call GetScanners API	A-7
Calling ExecCommand API to Demonstrate Beep the Beeper	A-8
Retrieve Asset Tracking Information from ExecCommand with the RSM_GET Method	A-9
Enable the UPC-A Attribute by Calling SET_ATTRIBUTE via ExecCommand	A-11
Capture Bar Code Data into an Application	A-12

Index**Quick Startup**

ABOUT THIS GUIDE

Introduction

This guide provides information about the Motorola Scanner Software Developer Kit (SDK) - an architectural framework providing a single programming interface across multiple programming languages and across multiple system environments for all scanners communication variants (such as IBMHID, SNAPI, SSI, HIDKB, Nixdorf Mode B, etc.).

Chapter Descriptions

Topics covered in this guide are as follows:

- [Chapter 1, INTRODUCTION TO THE MOTOROLA SCANNER SDK](#) provides an overview of the Motorola Scanner Software Developer Kit (SDK).
- [Chapter 2, INSTALLATION & CONFIGURATION](#) describes how to install Motorola Scanner SDK and its components on recommended platforms.
- [Chapter 3, MOTOROLA SCANNER SDK API](#) provides the set of APIs to interface with scanner devices.
- [Chapter 4, TEST UTILITIES & SOURCE CODE](#) provides information about testing and evaluation of the Motorola Scanner SDK's software components using the test utilities provided in the SDK.
- [Chapter 5, SAMPLE SOURCE CODE](#) provides information about how a developer uses the Motorola Scanner SDK.
- [Appendix A, WRITE SIMPLE APPLICATIONS USING CORE SCANNER APIS](#) provides a step by step guide to writing simple applications using CoreScanner APIs.

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 will 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 contact Motorola support for your region. Contact information is available at: <http://supportcentral.motorola.com>.

When contacting Mobility support, please have the following information available:

- Serial number of the unit
- Model number or product name
- Software type and version number.

Motorola responds to calls by E-mail, telephone or fax within the time limits set forth in support agreements.

If your problem cannot be solved by Motorola support, you may need to return your equipment for servicing and will be given specific directions. Motorola is not responsible for any damages incurred during shipment if the approved shipping container is not used. Shipping the units improperly can possibly void the warranty.

If you purchased your business product from a Motorola business partner, contact that business partner for support.

CHAPTER 1 INTRODUCTION TO THE MOTOROLA SCANNER SDK

Overview

The Motorola Scanner Software Developer Kit (SDK) defines an architectural framework providing a single programming interface across multiple programming languages (such as MS .NET, C++, Java) and across multiple system environments (such as Windows XP, Vista, Linux) for all scanners communication variants (such as IBMHID, SNAPI, HIDKB, Nixdorf Mode B, etc.).

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

With this SDK you can read bar codes, manage scanner configurations, capture images/videos and selectively choose a list of scanners on which to work. While one application is in one programming language using a scanner or a set of scanners, another application in a different language can be used differently within the same system environment.

For a list of the most commonly requested topics within this guide, see [Quick Startup](#) in the back of the guide.

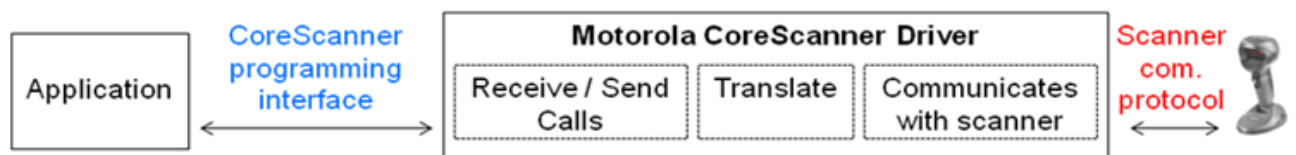


Figure 1-1 *Software Developer Framework*

The SDK can build an application with complete control of its scanner's capabilities.

- Data, Barcode
 - Simulation HID Keyboard output
 - OPOS/JPOS output
 - SNAPI output

- Command and Control
 - LED and Beeper Control
 - Aim Control
- Imaging
 - Capture / Transfer of images
 - Capture / Transfer of Video
- Remote Scanner Management
 - Asset Tracking
 - Device Configuration (Get, Set and Store Scanner attributes)
 - Firmware Upgrade
 - Scanner Communication Protocol Switching
 - Service to Automate Configuration / Firmware Upgrade Process.



NOTE For a complete list of attribute (parameter) numbers and their definitions, download the Attribute Data Dictionary (p/n 72E-149786-xx) from <http://MotorolaSolutions.com/WindowsSDK>. Attributes include configuration parameters, monitored data and asset tracking information.

Quick Startup

Overview	1-1
Operating systems / System requirements	2-2
Scanner model vs. Communication modes	1-6, 2-3
Block diagram of system	1-5
SDK Components & Installation details	2-1, 2-4
Components and folder paths	2-2, 2-9
Validate SDK installed properly	2-12, 4-6
OPOS / JPOS Drivers	2-1, 2-2, 2-7
WMI / Remote Scanner Management	2-2, 2-4, 2-7, 2-9
Test and sample utilities	4-2
Table of buttons and input fields	4-3
List of utility functionality	4-2
Bar code Data Display	A-12, A-11, 4-8, 4-9
One application connected to two scanners	1-9
Simulated HID Keyboard Output	1-6, 2-11, 4-3, 4-9
Discovery	4-6
Querying asset information	A-9, 4-6, 4-15, 4-17
Query and Set Parameters / Attributes	
Query values	4-15, 4-17, 4-18
Set Value (Device Configuration)	3-12, 4-19
Programming an ADF rule	4-18, 4-21
LED control	4-14, 4-14, 4-22
Beeper control	A-8, 4-13, 4-13, 4-22
Enable / disable a symbology	A-11, 4-20
Capturing an image	4-10
Capturing a video	4-10
Firmware Upgrade	4-24, 4-26
Host Variant Switching	4-24, 4-23
C++ sample application and source code	2-9, 4-2, 5-1
C# sample application and source code	2-9, 4-3, 5-1
Starter application using CoreScanner API	A-1
API overview	3-1
Create com object	5-1
Register for event	5-2, 4-6
Open	A-6, 3-2, 5-2
Get scanner	A-7, 3-3, 4-7, 5-3
Execute command	A-8, 4-15, 3-3, 5-3
List of Methods	3-12
Execute command asynchronously	3-4, 5-3
Close	3-4, 5-2
Attribute Data Dictionary (Index of Parameters)	4-1

FAQs

- Can multiple scanners be connected to the CoreScanner Driver?
 - Yes, multiple scanners can be connected simultaneously to one host running the CoreScanner driver.
- If two scanners are connected, can data be tracked by scanner ID?
 - Yes, if scanner X decodes a bar code 123, it will return to the application a data event consisting of 123 as the data label and the serial number as the scanner ID.
- Can multiple applications be connected to the CoreScanner Driver?
 - Yes, multiple applications can be connected simultaneously on one host running the CoreScanner driver. An application can register from a selection of event types (such as bar code, image, video or management). The application will receive the event information plus the originating scanner ID.
- Are the CoreScanner calls common across operating systems?
 - Yes. For example, the Open method's function signatures for C++ and Java are the same except for the platform specific data and return types (highlighted in yellow below).

JAVA

`int` ← `Open(long appHandle, short[] sfTypes, int lengthOfType, long status);`

`HRESULT` ← `Open(LONG appHandle, SAFEARRAY* sfTypes, SHORT lengthOfType, LONG* status);`

C++

Figure 1-2 Function Signatures for C++ and Java

Motorola Scanner SDK Architecture

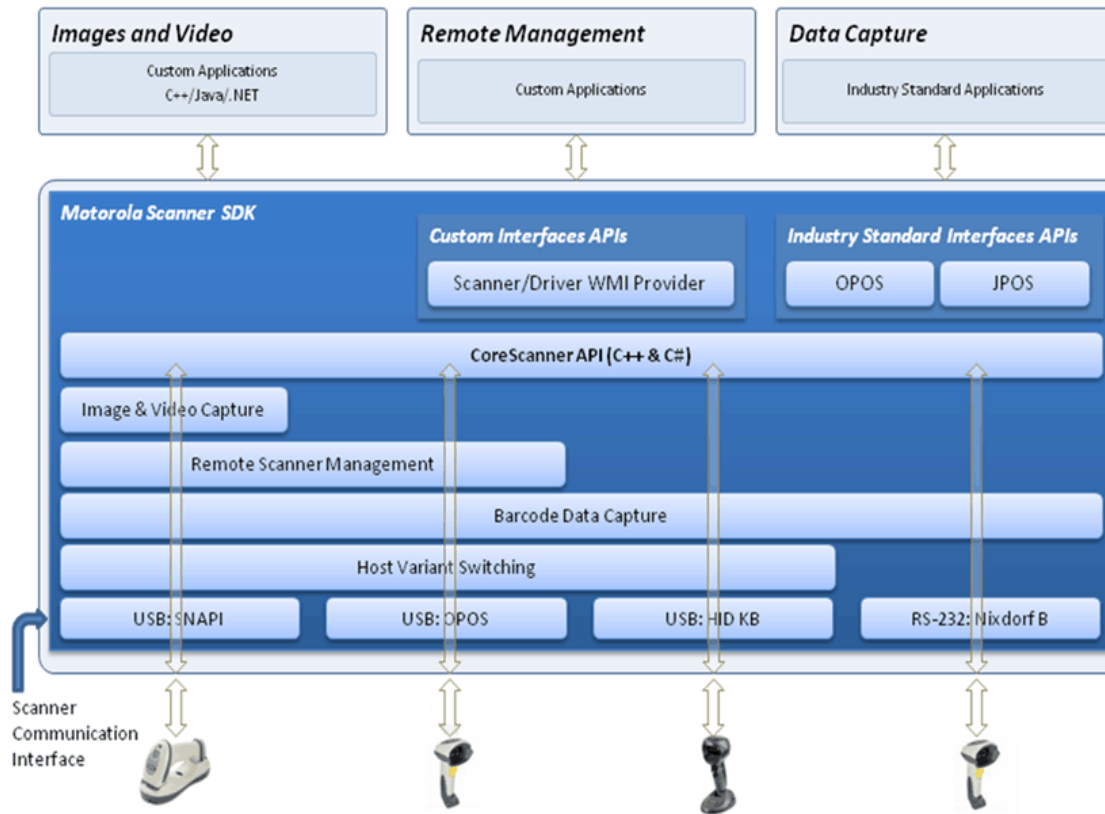


Figure 1-3 SDK Architecture

You can configure Motorola scanner devices to be operated in different host communication modes such as USB SNAPI, USB OPOS, USB HID Keyboard and RS232 Nixdorf Mode B. Device feature support varies depending on communication mode but all modes support bar code scanning. Refer to the Product Reference Guide of a specific scanner for the bar codes to set its supported host communication modes.

Using the Motorola Scanner SDK, you can switch between supported host communication modes by calling the host variant switching command programmatically. This is useful when the device is in a less feature supportive mode and some advanced functionality is required but not supported by the current communication mode. The scanner can be switched to a feature rich mode and commands executed before switching the scanner back to the previous mode.

For example, you want to disable the UPC-A symbology but the device is in USB HID Keyboard mode. If the mode is supported by the scanner, you may switch to USB SNAPI or USB OPOS, set UPC-A to be disabled permanently and then switch the scanner back to USB HID Keyboard mode. See [Table 1-1 on page 1-6](#) for more information.

Table 1-1 illustrates scanner capabilities supported by each communication mode. Refer to the specifications of a device for its ability to support of each communication mode.

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

Capabilities	USB SNAPI	USB OPOS	USB HID Keyboard	RS232 Nixdorf B
Data	Supported	Supported	Supported	Supported
Host Variant Switching	Supported	Supported	Supported	Not Available
Management	Supported	Supported	Not Available	Not Available
Image & Video	Supported	Not Available	Not Available	Not Available
Simulated HID Keyboard Output *	Supported	Supported	Not Applicable	Supported *

***Advanced Data Formatting (ADF) is not supported when using Simulated HID Keyboard Output.**

Simulated HID Keyboard Output is a feature enabling scanners in USB SNAPI or USB OPOS mode to emulate HID Keyboard keystrokes to a host system for scanned data labels. It sends the content of the scanned data label as HID Keyboard keystrokes thus emulating USB HID Keyboard scanner mode.

Multiple Scanner Device Identification Methodology For Applications

The Motorola Scanner SDK supports multiple scanner devices to any application that runs on top of CoreScanner APIs. Each scanner device is shown to the user application by a unique scanner identification number. The Scanner ID is a numeric value assigned to each connected device so there cannot be multiple scanner devices holding the same Scanner ID.

Asset tracking information like model number, serial number, current firmware version and date of manufacture are available if the scanner and its current host mode support the management feature.

For example, in some modes like USB HID Keyboard, you do not see asset tracking information but the same scanner device will show you such information when it is in USB OPOS or USB SNAPI mode.

The format of device asset tracking information can follow different naming conventions for device model, serial number or current firmware version. For example, the length of a serial number for DS6707 and DS9808 scanners can be different.

How Multiple Applications Access Multiple Scanners From Scanner SDK

The Motorola Scanner SDK supports multiple applications accessing multiple scanner devices connected to the host at the same time.

As described previously, a Scanner ID uniquely identifies a connected scanner device to all applications. A Scanner ID will be consistent among all applications for one SDK instance. If the CoreScanner service or the host machine is restarted, a device may be assigned a different Scanner ID but it will be unique and referenced by all applications.

Three Applications Connected To One Scanner

Figure 1-4 illustrates how multiple applications communicate with multiple scanner devices.

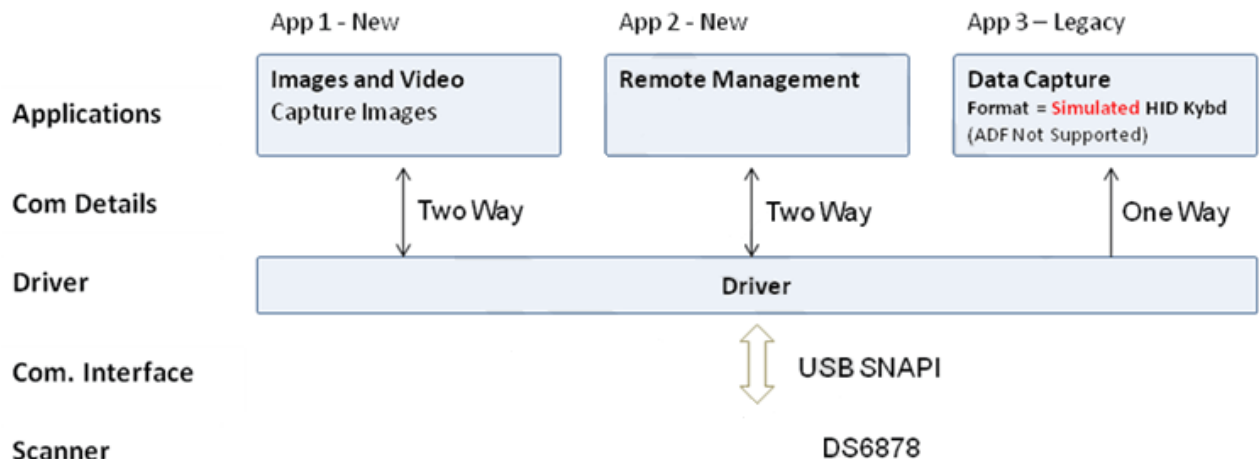


Figure 1-4 Three Applications Connected To One Scanner

Implementation Details

- Three applications are connected to one scanner.
- App 1 & App 2 support bi-directional (two way) communication with the scanner.
- Legacy App 3 supported by driver converting SNAPI data into HID format.

Three Applications Connected To Two Scanners

Figure 1-4 illustrates how multiple applications communicate with multiple scanner devices.

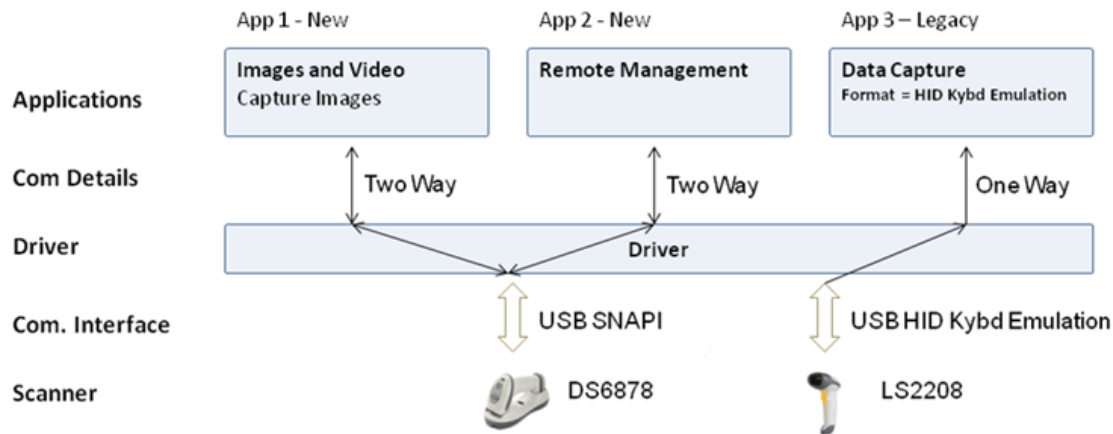


Figure 1-5 Three Applications Connected To Two Scanners

Implementation Details

- Three applications are connected to two scanners.
- App 1 and App 2 support bi-directional (two way) communication with the DS6878.
- Legacy App 3 receives HID keyboard emulation data from the LS2208.

Many-to-Many Application Device Usage

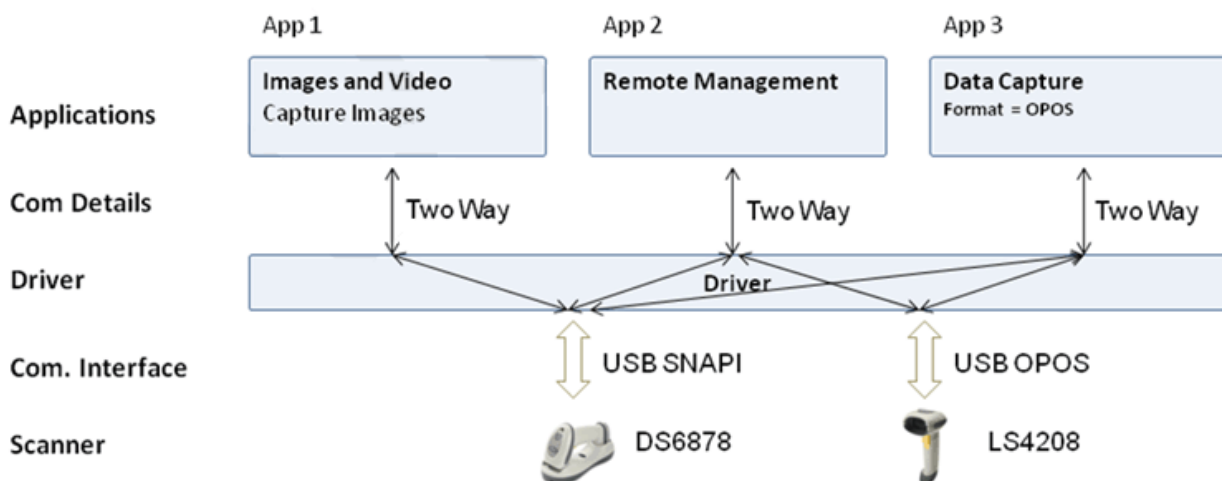


Figure 1-6 Many-to-Many Application Device Usage

Implementation Details

- App 1 performs image capture with the DS6878.
- App 2 can remotely manage both the DS6878 and LS4208.
- App 3 receives OPOS data from both the DS6878 and LS4208.

One Application Connected to Two Scanners

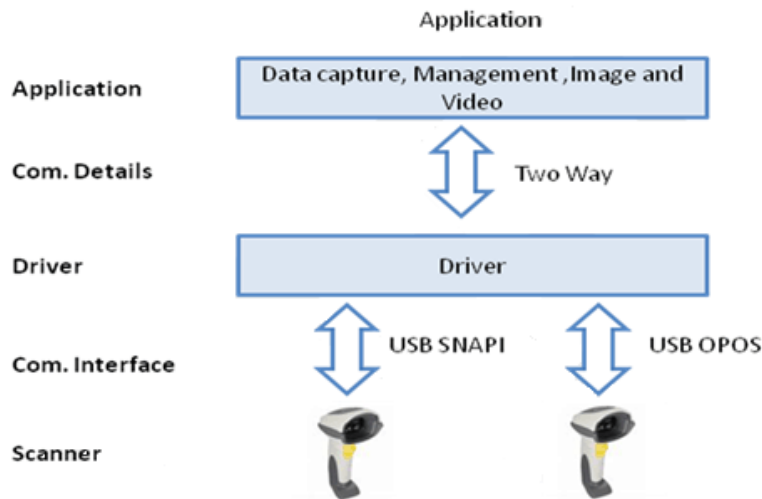


Figure 1-7 One Application Connected to Two Scanners

Implementation Details

- 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 responses from the scanners consist of the scanner device details (ScannerID, serial number, model number, etc.) identifying the device that sent the response.

For example, a bar code event for a scanned label is shown below. The scanned data label arrives with a unique ScannerID and the scanner's model number and the 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 Motorola Scanner SDK and its components on recommended platforms.

✓ **NOTE** See [System Requirements on page 2-2](#) for supported platforms.

The SDK installation package includes support for:

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

This section covers installation and configuration instructions.

✓ **NOTE** Uninstall any previous Motorola, Symbol or 3rd party drivers or SDKs installed on your system which communicate with Motorola Scanner Devices before installing the Motorola Scanner SDK. This includes but is not limited to Motorola and Symbol supplied OPOS, JPOS and SNAPI drivers.

For a list of the most commonly requested topics within this guide, see [Quick Startup](#) in the back of the guide.

✓ **NOTE** For a complete list of attribute (parameter) numbers and their definitions, download the Attribute Data Dictionary (p/n 72E-149786-xx) from <http://MotorolaSolutions.com/WindowsSDK>. Attributes include configuration parameters, monitored data and asset tracking information.

SDK Components

The SDK installation package includes following components.

- Motorola Scanner SDK Core components and drivers (COM API, Imaging drivers)
- Scanner OPOS Driver - Software
- Remote Management Components
 - Scanner WMI Provider
 - Driver WMI Provider
- Web Link to latest Developer's Guide - Document(s)
- Test & Sample utilities with Source code packages
 - Motorola Scanner SDK Sample Application (C++)
 - Motorola Scanner SDK Sample Application (Microsoft® C# .NET)
 - Scanner OPOS Driver Test Utility (C++)
 - Scanner JPOS Driver Test Utility (Java)
 - Scanner WMI Provider Test Utility (Microsoft® C# .NET)
 - Driver WMI Provider Test Utility (Microsoft® C# .NET).

The SDK installation package installs its components to the following default location:

C:\Program Files\Motorola Scanner\.

System Requirements

Supported Operating Systems

Table 2-1 *Supported Operating Systems*

Motorola Scanner SDK Win32 Installation Package	
Microsoft® Windows XP SP3 (32bit)	ScannerSDKv1[1].00.xxxx.exe
Microsoft® Windows 7 (32bit)	ScannerSDKv1[1].00.xxxx.exe

Recommended minimum hardware requirement: x86 Computer with 512Mb RAM.

Scanner Models Versus Communication Modes

[Table 2-2](#) shows the supported communication modes for each scanner model.

Table 2-2 Scanner Model Versus Communication Interface

Scanner Model	Supported Host Variants				Capabilities				
	USB			RS-232					
	SNAPI	OPOS/IBM Hand Held	Keyboard emulation mode	Nixdorf Mode B	Barcode Scanning	Image Capture	Video Capture	Firmware Update	
								USB OPOS	USB Bulk*
DS6707	X	X	X	X	X	X	X	X	X
DS9808	X	X	X	X	X	X	-	X	X
DS6878	X	X	X	X	X	X	X	X	-
DS3408		X	X	X	X	-	-	X	-
DS6708		X	X	X	X	-	-	X	-
DS3407	X	X	X	X	X	X	X	-	-
LS4208	-	X	X	X	X	-	-	X	-
LS2208	-	X	X	X	X	-	-	-	-
LS4278	-	X	X	X	X	-	-	X	-
LS3578	-	X	X	X	X	-	-	X	-
LS9208	-	X	X	X	X	-	-	X	-
LS3408	-	X	X	X	X	-	-	X	-
LS7708	-	X	X	X	X	-	-	-	-
LS7808	-	X	X	X	X	-	-	-	-
DS4208	X	X	X	X	X	-	-	X	X
DS9208	X	X	X	X	X	X	-	X	X

* USB Bulk = USB SNAPI communication mode.

Installing the SDK

Download the Motorola Scanner SDK setup program from <http://MotorolaSolutions.com/WindowsSDK>.

There are two options for installing the Motorola Scanner SDK on a system.

- Typical installation - Loads all components in the installation package.
- Custom installation - Provides the ability to change the default selection of components.

If you install components such as OPOS, JPOS or WMI provider (remote management), the installer automatically installs sample programs and test utilities related to those components.

To download the appropriate OPOS, JPOS and WMI Developer's Guides go to:

<http://MotorolaSolutions.com/WindowsSDK>.

Step-by-Step Installation Instructions

1. Execute the setup program. The installation process checks for CoreScanner drivers on the target machine. If the driver package is not present or outdated, clicking **Install** adds updated drivers before installing the Scanner SDK package.

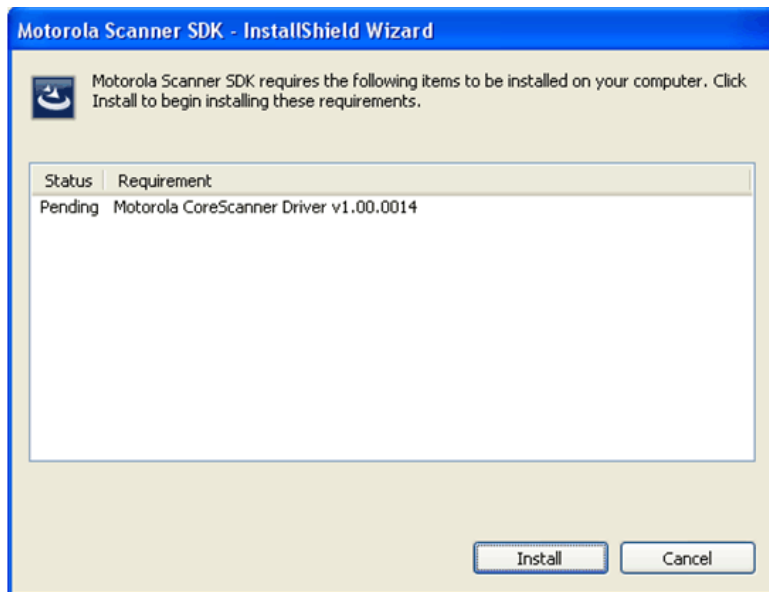


Figure 2-1 Prerequisite Check And Install

2. Installation continues once the prerequisite drivers are installed on the machine.

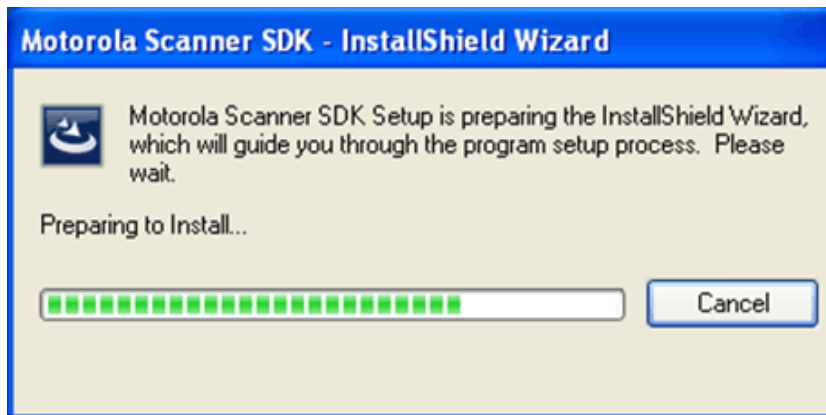


Figure 2-2 Initial Window

3. Click **Next** on the *Welcome* screen.

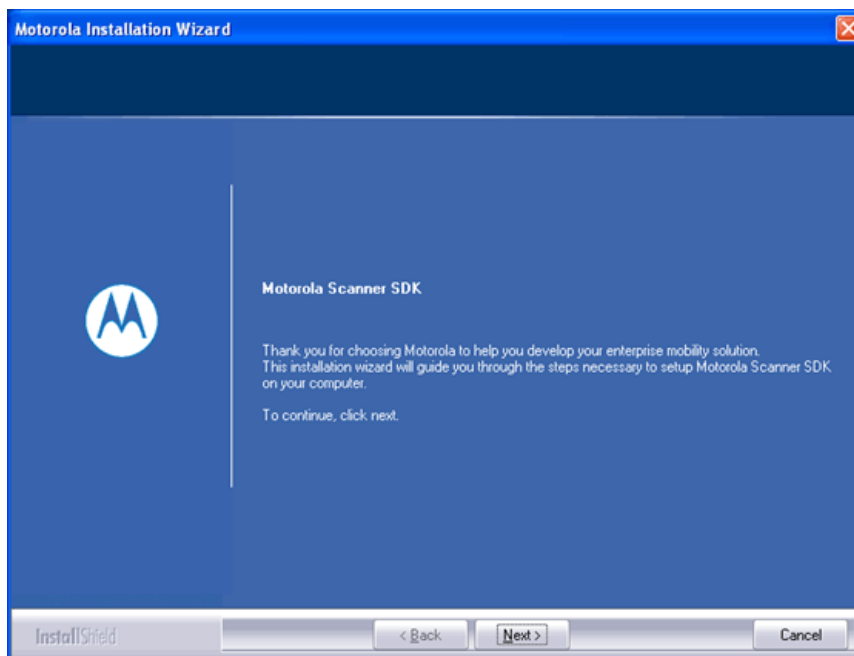


Figure 2-3 Welcome Screen

4. Review the license agreement and click **Yes** to accept.

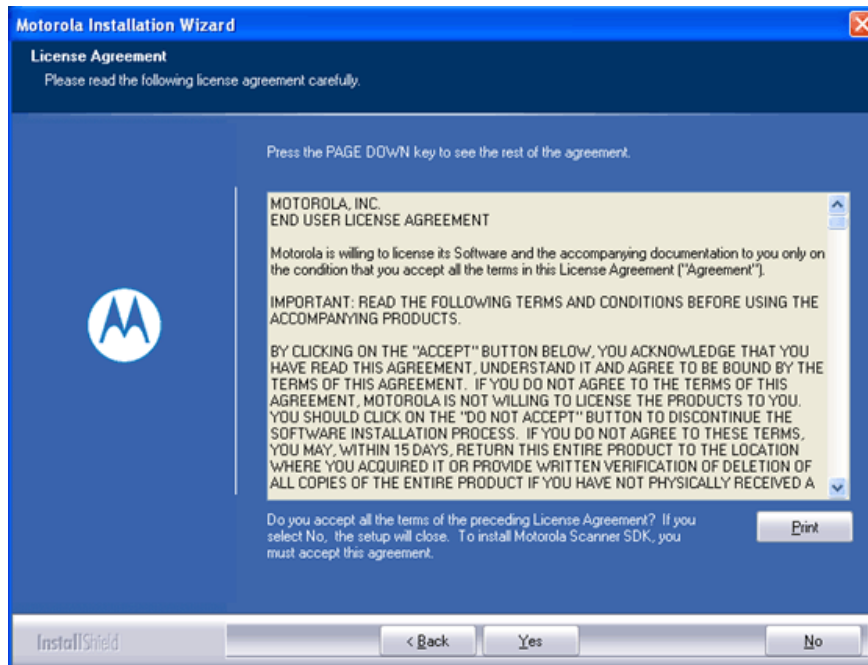


Figure 2-4 License Agreement

5. Select the *Setup Type*.

The user is prompted with two installation options:

- Complete - The installation package installs all components.
- Custom - The installation package gives the option to select which components are loaded during the installation process. The user is prompted to select components from the available list.

The user can select the destination folder by clicking **Browse** and selecting the drive and folder in which to install the Motorola Scanner SDK. However, common components are placed in designated locations for consistency with other SDK users.

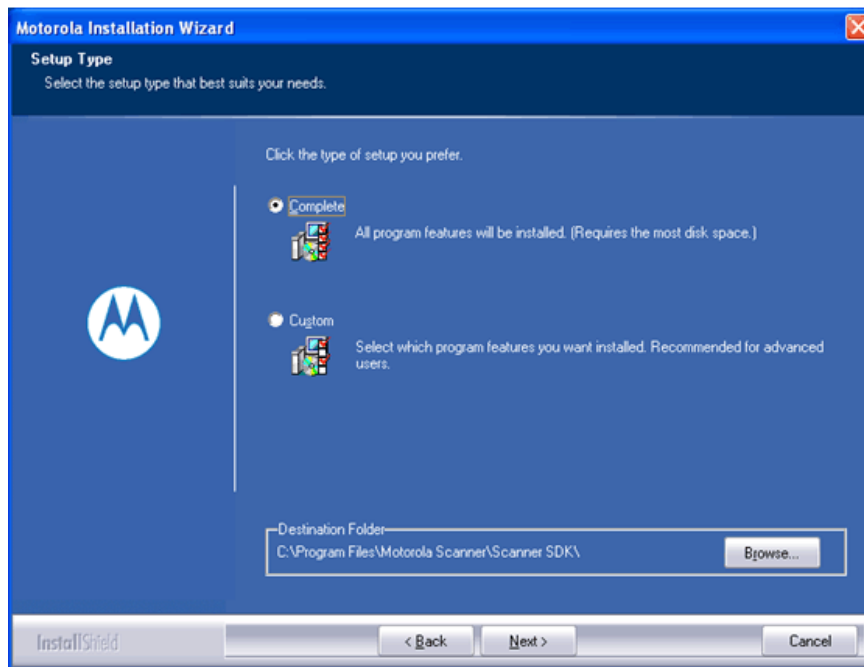


Figure 2-5 Setup Type

6. Select features. The user is prompted to select features to be installed from the available components list.

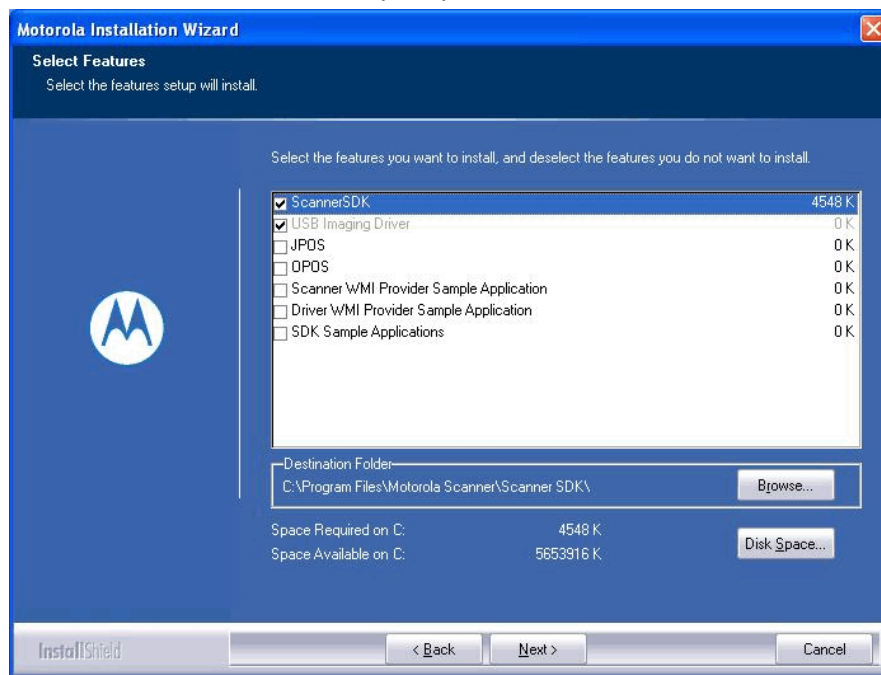


Figure 2-6 Select Features



NOTE Scanner SDK and USB imaging drivers are common components and are installed automatically.

7. Wait for the installation to complete.

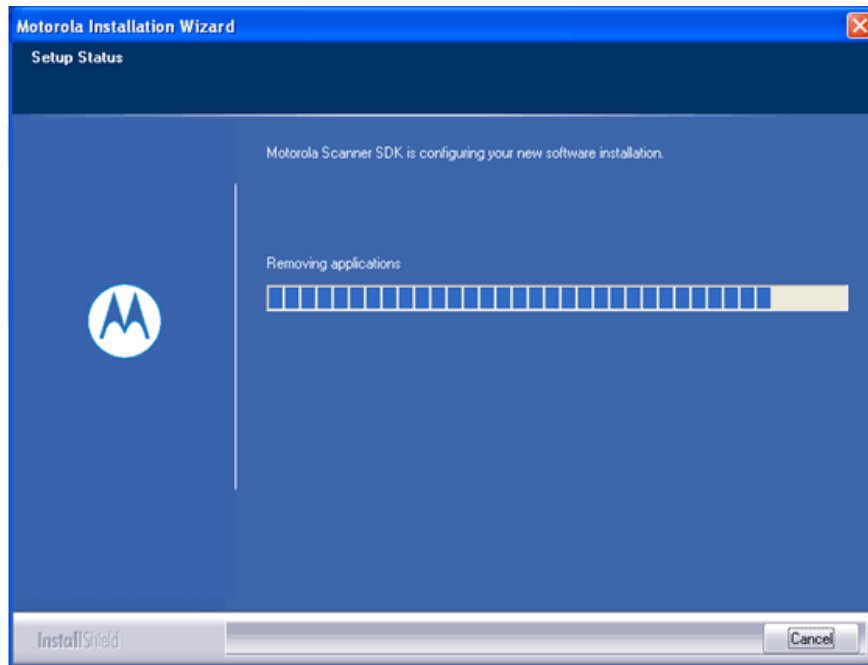


Figure 2-7 *Installation Progress*

8. Installation complete.

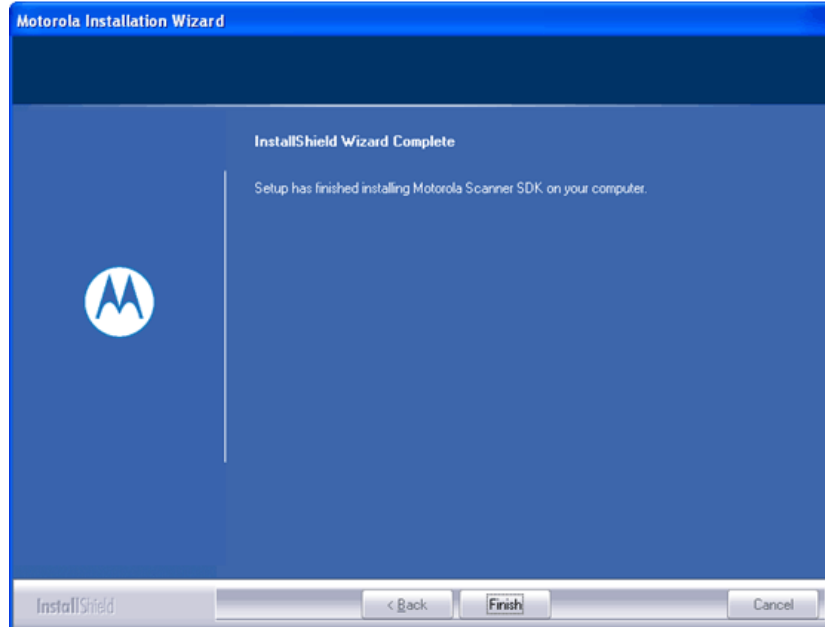


Figure 2-8 *Installation Complete*

Installed Components

[Table 2-3](#) lists the components installed.

Table 2-3 Motorola Scanner SDK Components

Component	File	Description	Installation Path
Scanner Driver	CoreScanner.exe	Scanner Driver/COM Server	.Common
Scanner Driver	SNAPITrans.dll	Transport Component	.Common
Scanner Driver	USBHIDKBTrans.dll	Transport Component	.Common
Scanner Driver	NIXBTrans.dll	Transport Component	.Common
Scanner Driver	IBMHIDTrans.dll	Transport Component	.Common
WMI Providers	ScannerService.exe	WMI Provider Services	.Common
WMI Providers	symbscnr.dll	WMI Instance Providers	.Common
WMI Providers	ScannerWMITest.sln	Scanner WMI Sample Application	\Scanner SDK\wmipprovider_scanner\Sample Applications\src
WMI Providers	RSMDriverProviderService.exe	WMI Provider Services	.Common
WMI Providers	RSMDriverProvider.dll	WMI Instance Providers	.Common
WMI Providers	symbscnr.mof	Managed Object Format file for WMI and CIM	.Common
WMI Providers	RSMDriverProvider.mof	Managed Object Format file for WMI and CIM	.Common
WMI Providers	DriverWMITest.sln	Driver WMI Sample Application	\Scanner SDK\wmipprovider_driver\Sample Applications\src
Configuration	config.xml	Scanner Driver Configuration File	.Common
Scanner Driver	SnapInstApp.exe	Imaging Driver Installer	.Common\ Usb imaging driver
Scanner Driver	SNAPIImg.sys	Imaging Driver	.Common\ Usb imaging driver \Drivers
Scanner Driver	snapiimg.inf	Imaging Driver inf file	.Common\ Usb imaging driver \Drivers
Scanner Driver	SNAPIImg.cat	Imaging Driver Catalog	.Common\ Usb imaging driver \Drivers
SDK C++ Sample App source code	SampleApp_CPP.sln	SDK C++ Sample Application and source projects	.Scanner SDK\ Scanner SDK\Sample Applications\src
SDK C# Sample App source code	SampleApp_CSharp.sln	SDK C++ Sample Application and source projects	.Scanner SDK\ Scanner SDK\Sample Applications\src

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

Component	File	Description	Installation Path
OPOS	OPOSScanner.ocx	OPOS Scanner Control	.\ Scanner SDK\OPOS\bin
OPOS	STIOPOS.dll	OPOS Service Object	.\ Scanner SDK\OPOS\bin
OPOS	TestScan.sln	OPOS Sample application source project	.\ Scanner SDK\OPOS\Sample Applications\src
JPOS	CSJPOS.dll	JPOS JNI Layer for CoreScanner API	.\ Scanner SDK\JPOS\bin
JPOS	POStest	JPOS Sample Application	.\ Scanner SDK\JPOS\Sample Applications\src

Configuration

The Motorola Scanner SDK is capable of communicating with scanners connected to serial ports through serial host mode. The SDK will not open any serial port without user consent to prevent other devices from being interfered with by Scanner SDK commands. Users can configure each serial port by adding entries in the config.xml file as described in this section.

Enter the correct port id, baud rate and the serial host mode corresponding to the serial port scanner.

Simulated HID Keyboard Output

The Motorola Scanner SDK is capable of configuring a scanner for use as a keyboard emulator (as in HID KB Emulation mode) while in USB SNAPi, USB OPOS or RS232 Nixdorf Mode B communication modes. This Simulated HID Keyboard Output functionality can be configured by changing the value of the <ENABLE> tag under <HID_KB_PUMP_SETTINGS> to "1" in the config.xml that resides in the same location as CoreScanner.exe. By default, the language locale of the simulated keyboard is English. Currently, only the English and French languages are supported. To set the language locale to French, change the value of the <LOCALE> tag under <HID_KB_PUMP_SETTINGS> to "1".

Sample config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<config>

  <serial_mode_settings>
    <port id="1" baud="9600" name="NIXMODB"/>
    <port id="2" baud="9600" name="SSI"/>
    <port id="3" baud="19200" name="SSI"/>
  </serial_mode_settings>

  <HID_KB_PUMP_SETTINGS>
    <LOCALE>0</LOCALE>      <!-- ENGLISH=0, FRENCH=1 -->
    <ENABLE>0</ENABLE>      <!-- ENABLED=1, DISABLED=0 -->
  </HID_KB_PUMP_SETTINGS>

</config>
```



- NOTES**
1. Refer to the specific scanner Product Reference Guide for supported serial port parameter settings.
 2. Simulated HID Keyboard Output settings can be temporarily changed by an application using the CoreScanner API commands KEYBOARD_EMULATOR_ENABLE and KEYBOARD_EMULATOR_SET_LOCALE. To make permanent changes to these settings that will remain persistent over a reboot of the host machine, the Config.xml file must be manually edited. Changes to Config.xml will take effect only after the CoreScanner service is restarted.
 3. When using the language locale with Simulated HID Keyboard Output, the user may need to change the input language of the application receiving keyboard input to match the language specified in Config.xml.
 4. After a Windows logout/login or user switch, manually restart the CoreScanner, RSM Driver Provider and Symbol Scanner Management services to ensure correct operation.

Basic Installation Verification

You can perform a basic inspection on your system process list to verify a successful installation of the Motorola Scanner SDK.

✓ **NOTE** This is simple verification of the operation of the Motorola Scanner SDK. See [How to Verify Scanner SDK Functionality on page 4-6](#) for more advanced SDK testing.

The following instructions guide you through a simple check of the Scanner SDK's operation.

1. Right click on the *Windows Task Bar* and select *Task Manager*.

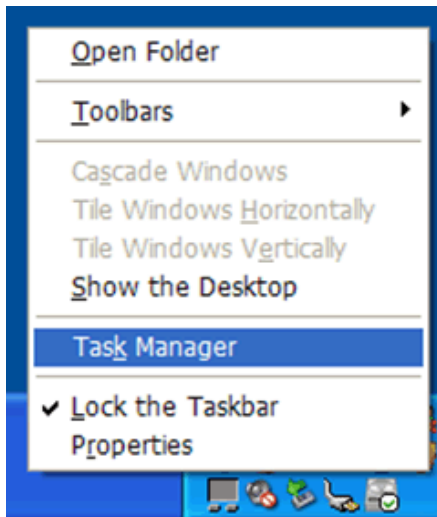


Figure 2-9 Task Bar Selection of Task Manager

2. Under the *Processes* tab, find the *CoreScanner.exe* in the *Image Name* list under.

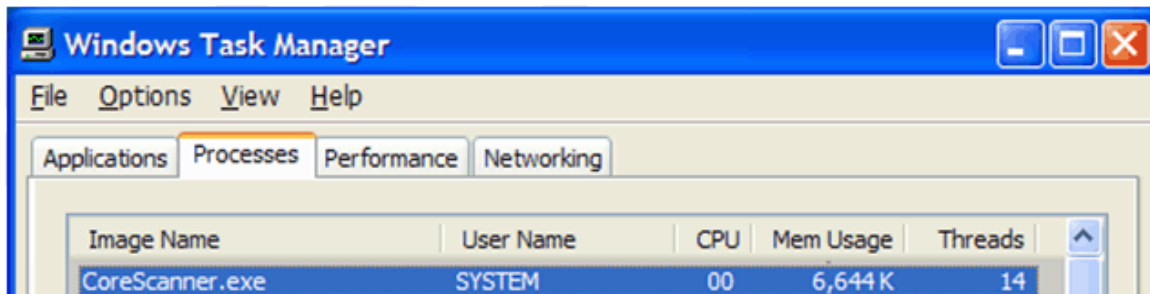


Figure 2-10 CoreScanner.exe on Task Manager

3. The appearance of "CoreScanner.exe" in the Processes list indicates a successful installation.

CHAPTER 3 MOTOROLA SCANNER SDK API

Overview

The Motorola Scanner SDK provides an easy to use yet powerful and extendible set of API commands to interface with scanner devices. The API commands include:

- Open
- GetScanners
- ExecCommand
- ExecCommandAsync
- Close.

Once the SDKs Open and GetScanners commands are invoked and the list of connected scanners is retrieved, all other methods will execute through the ExecCommand and ExecCommandAsync commands. This is a user friendly approach and easy to code in terms of day-to-day programming.

With the evolution of the SDK's capabilities, it is easier to increase the number of methods rather than increase the number of API commands. The benefit to the user is that, once you have the system up and running, a new method is just an additional operation to the existing code.

In addition to the commands above, the Motorola Scanner SDK supports seven types of events:

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

See [Appendix A, WRITE SIMPLE APPLICATIONS USING CORE SCANNER APIS](#) for a starter example of an application illustrating the Motorola Scanner SDK API. For a table listing the most commonly requested topics within this guide, see [Quick Startup](#) in the back of the guide.



NOTE For a complete list of attribute (parameter) numbers and their definitions, download the Attribute Data Dictionary (p/n 72E-149786-xx) from <http://MotorolaSolutions.com/WindowsSDK>. Attributes include configuration parameters, monitored data and asset tracking information.

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.

```
HRESULT STDMETHODCALLTYPE Open(
    /* [in] */ LONG reserved,
    /* [in] */ SAFEARRAY * sfTypes,
    /* [in] */ SHORT lengthOfTypes,
    /* [out] */ LONG *status) = 0;
```

reserved - Reserved argument. Set to 0.

sfTypes - Selects the types of scanners requested for use with the API.

Table 3-1 Values for *sfTypes*

Code	Value	Scanner Category
SCANNER_TYPES_ALL	1	All Scanners
SCANNER_TYPES_SNAPI	2	SNAPI Scanners
SCANNER_TYPES_IBMhid	6	IBM Hand Held Scanners (USB OPOS)
SCANNER_TYPES_NIXMODB	7	Nixdorf Mode B scanners (RS232)
SCANNER_TYPES_HIDKB	8	USB HID Keyboard emulation scanners

lengthOfTypes - Number of elements or the size of *sfTypes* array

status - Return value for the command

Return Values

0 - Success.

Any other value - See [Error and Status Codes on page 3-32](#).

GetScanners

Gets a list of scanners of the requested types that are connected at any time. This command should be invoked after the Open command.

```
HRESULT STDMETHODCALLTYPE GetScanners(
    /* [out] */ SHORT *numberOfScanners,
    /* [out][in] */ SAFEARRAY * sfScannerIDList,
    /* [out] */ BSTR *outXML,
    /* [out] */ LONG *status) = 0;
```

numberOfScanners - Number of connected scanners of requested type(s).

sfScannerIDList - 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 3-32](#).

ExecCommand

Provides synchronous execution of a method via an opcode.

```
HRESULT STDMETHODCALLTYPE ExecCommand(
    /* [in] */ LONG opcode,
    /* [in] */ BSTR *inXML,
    /* [out] */ BSTR *outXML,
    /* [out] */ LONG *status) = 0;
```

opcode - Method to be executed. See [Table 3-9 on page 3-12](#) 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 3-32](#).

ExecCommandAsync

Provides asynchronous execution of a method via an opcode. Any response data is retrieved as CommandResponseEvents. See [CommandResponseEvent on page 3-11](#).

```
HRESULT STDMETHODCALLTYPE ExecCommandAsync(
    /* [in] */ LONG opcode,
    /* [in] */ BSTR *inXML,
    /* [out] */ LONG *status) = 0;
```

opcode - Method to be executed. See [Table 3-9 on page 3-12](#) 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 3-32](#).

Close

Closes the application instance through the CoreScanner service.

```
HRESULT STDMETHODCALLTYPE Close(
    /* [in] */ LONG appHandle,
    /* [out] */ LONG *status) = 0;
```

appHandle - Application ID.

status - Return value for the command.

Return Values:

0 - Success.

Any other value - See [Error and Status Codes on page 3-32](#).

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 for this purpose (see [Table 3-9 on page 3-12](#)).

ImageEvent

Triggered when an imaging scanner captures images in image mode.

Syntax

```
void OnImageEvent (
    short eventType
    int size
    short imageFormat,
    ref object sfimageData,
    ref 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	Triggered when complete image captured
IMAGE_TRAN_STATUS	2	Triggered when image error or status

size

Size 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.

Syntax

```
void OnVideoEvent(
    short eventType,
    int size,
    ref object sfvideoData,
    ref string pScannerData)
```

Parameters

eventType

Type of video event received (see [Table 3-4](#)).

size

Size of video data buffer.

sfvideoData

Video data buffer.

pScannerData

Reserved parameter: always returns an empty string.

Table 3-4 Video Event Types

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

BarcodeEvent

Triggered when a scanner captures bar codes.

Syntax

```
void OnBarcodeEvent(
    short eventType,
    ref string pscanData)
```

Parameters

eventType

Type of bar code event received (see [Table 3-4](#)).

Table 3-5 Bar Code Event Types

Event Type	Value	Description
SCANNER_DECODE_GOOD	1	Triggered when a decode is successful.

pscanData

Bar code string that contains information about the scanner that triggered the bar code event including data type, data label and raw data of the scanned bar code.

```
<?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>
```

[Table 3-6](#) lists the data types of scanned bar codes.

Table 3-6 Bar Code Data Types

ID	Bar Code Type
1	Code 39
2	Codabar
3	Code 128
4	Discrete (Standard) 2 of 5
5	IATA
6	Interleaved 2 of 5
7	Code 93

Table 3-6 *Bar Code Data Types (Continued)*

ID	Bar Code Type
8	UPC-A
9	UPC-E0
10	EAN-8
11	EAN-13
12	Code 11
13	Code 49
14	MSI
15	EAN-128
16	UPC-E1
17	PDF-417
18	Code 16K
19	Code 39 Full ASCII
20	UPC-D
21	Code 39 Trioptic
22	Bookland
23	Coupon Code
24	NW-7
25	ISBT-128
26	Micro PDF
27	DataMatrix
28	QR Code
29	Micro PDF CCA
30	PostNet US
31	Planet Code
32	Code 32
33	ISBT-128 Con
34	Japan Postal
35	Australian Postal
36	Dutch Postal
37	MaxiCode
38	Canadian Postal

Table 3-6 *Bar Code Data Types (Continued)*

ID	Bar Code Type
39	UK Postal
40	Macro PDF
48	RSS-14
49	RSS Limited
50	RSS Expanded
55	Scanlet
72	UPC-A + 2 Supplemental
73	UPC-E0 + 2 Supplemental
74	EAN-8 + 2 Supplemental
75	EAN-13 + 2 Supplemental
80	UPC-E1 + 2 Supplemental
81	CCA EAN-128
82	CCA EAN-13
83	CCA EAN-8
84	CCA RSS Expanded
85	CCA RSS Limited
86	CCA RSS-14
87	CCA UPC-A
88	CCA UPC-E
89	CCC EAN-128
90	TLC-39
97	CCB EAN-128
98	CCB EAN-13
99	CCB EAN-8
100	CCB RSS Expanded
101	CCB RSS Limited
102	CCB RSS-14
103	CCB UPC-A
104	CCB UPC-E
105	Signature Capture
113	Matrix 2 of 5

Table 3-6 Bar Code Data Types (Continued)

ID	Bar Code Type
114	Chinese 2 of 5
136	UPC-A + 5 supplemental
137	UPC-E0 + 5 supplemental
138	EAN-8 + 5 supplemental
139	EAN-13 + 5 supplemental
144	UPC-E1 + 5 Supplemental
154	Macro Micro PDF

PNPEvent

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

Syntax

```
void OnPNPEvent(
    short eventType,
    ref 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	Triggered when a Motorola Scanner is attached.
SCANNER_DETACHED	1	Triggered when a Motorola Scanner is detached.

ppnpData

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

ScanRMDEvent

Receives RMD Events when updating firmware of the scanner.

Syntax

```
void OnScanRMDEvent(
    short eventType,
    ref string prmdData)
```

Parameters

eventType

Type of the RMD event received (see [Table 3-8](#)).

prmdData

ScanRMD information string containing the data of event. (See [Firmware Upgrade Scenarios on page 4-24](#) for more details on this string.)

Table 3-8 RMD Event Types

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

CommandResponseEvent

Received after an asynchronous command execution (ExecCommandAsync).

Syntax

```
void OnCommandResponseEvent(
    short status,
    ref string prspData)
```

Parameters

status

Status of the executed command. (See [Error/Status Codes on page 3-32](#).)

prspData

CommandResponse information string that contains the outXML of the executed command.

Methods Invoked Through ExecCommand Or ExecCommandAsync

Table 3-9 *List of Methods*

Description	Method	Value
Scanner SDK Commands	GET_VERSION	1000
	REGISTER_FOR_EVENTS	1001
	UNREGISTER_FOR_EVENTS	1002
Scanner Access Control Command	CLAIM_DEVICE	1500
	RELEASE_DEVICE	1501
Scanner Common Command	ABORT_MACROPDF	2000
	ABORT_UPDATE_FIRMWARE	2001
	AIM_OFF	2002
	AIM_ON	2003
	FLUSH_MACROPDF	2005
	DEVICE_LED_OFF	2009
	DEVICE_LED_ON	2010
	DEVICE_PULL_TRIGGER	2011
	DEVICE_RELEASE_TRIGGER	2012
	SCAN_DISABLE	2013
	SCAN_ENABLE	2014
	SET_PARAMETER_DEFAULTS	2015
	DEVICE_SET_PARAMETERS	2016
	SET_PARAMETER_PERSISTANCE	2017
	DEVICE_BEEP_CONTROL	2018
	REBOOT_SCANNER	2019
Image Commands	DEVICE_CAPTURE_IMAGE	3000
Video Commands	DEVICE_CAPTURE_VIDEO	4000

Table 3-9 *List of Methods (Continued)*

Description	Method	Value
Scanner Management Commands	ATTR_GETALL	5000
	ATTR_GET	5001
	ATTR_GETNEXT	5002
	ATTR_SET	5004
	ATTR_STORE	5005
	GET_DEVICE_TOPOLOGY	5006
	START_NEW_FIRMWARE	5014
	UPDATE_FIRMWARE	5016
Serial Scanner Commands	DEVICE_SET_SERIAL_PORT_SETTINGS	6101
Other Commands	DEVICE_SWITCH_HOST_MODE	6200
Keyboard Emulator Commands	KEYBOARD_EMULATOR_ENABLE	6300
	KEYBOARD_EMULATOR_SET_LOCALE	6301
	KEYBOARD_EMULATOR_GET_CONFIG	6302

Examples: Using the Methods



NOTE The inXML segments that follow are only examples. inXMLs must be customized by the programmer based on each user's requirements.

GET_VERSION

Description: Gets the version of CoreScanner Driver

Asynchronous supported: No

Supported Scanner Communication Protocols: N/A

inXml:

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

outXml

```
<?xml version="1.0" encoding="UTF-8"?>
<outArgs>
  <arg-xml>
    <arg-string>01.00.00</arg-string>
  </arg-xml>
</outArgs>
```

Version of the CoreScanner driver

REGISTER_FOR_EVENTS

Description:

Register for API events described [API Events beginning on page 3-5](#)

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 3-10](#) lists the Event IDs for the inXML code above.**Table 3-10** 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

Description:

Unregister from API events described in [API Events beginning on page 3-5](#)

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>

```

outXml:

null

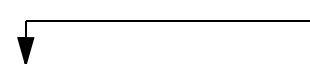
CLAIM_DEVICE

Description: Claim a specified device

Asynchronous supported: No

Supported Scanner Communication Protocols: IBM Handheld, SNAPI, HID Keyboard, NIXDORF Mode B

inXml:



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

outXml: null

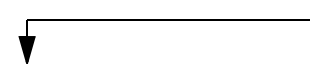
RELEASE_DEVICE

Description: Release a specified device

Asynchronous supported: No

Supported Scanner Communication Protocols: IBM Handheld, SNAPI, HID Keyboard, NIXDORF Mode B

inXml:



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

outXml: null

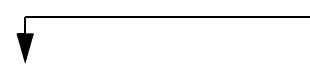
ABORT_MACROPDF

Description: Abort MacroPDF of a specified scanner

Asynchronous supported: No

Supported Scanner Communication Protocols: SNAPI

inXml:



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

outXml: null

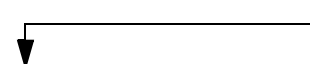
ABORT_UPDATE_FIRMWARE

Description: Abort Firmware updates process of a specified scanner while it is progressing

Asynchronous supported: No

Supported Scanner Communication Protocols: IBM Handheld, SNAPI

inXml:



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

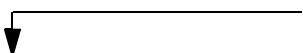
outXml: null

AIM_OFF

Description: Turn off the aiming of a specified scanner

Asynchronous supported: No

Supported Scanner Communication Protocols: SNAPI

inXml:  Specified Scanner ID

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

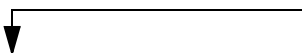
outXml: null

AIM_ON

Description: Turn on the aiming of a specified scanner

Asynchronous supported: No

Supported Scanner Communication Protocols: SNAPI

inXml:  Specified Scanner ID

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

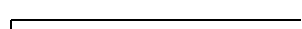
outXml: null

FLUSH_MACROPDF

Description: Flush MacroPDF of a specified scanner

Asynchronous supported: No

Supported Scanner Communication Protocols: SNAPI

inXml:  Specified Scanner ID

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

outXml: null

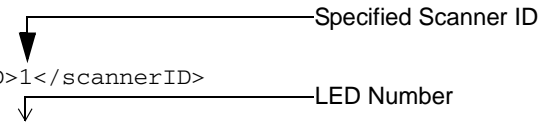
DEVICE_LED_OFF

Description: Turn off a given LED* of a specified scanner

Asynchronous supported: N/A

Supported Scanner Communication Protocols: IBM Handheld, SNAPi

inXml:



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

outXml: null

* Refer to the Product Reference Guide of the scanner for supported LED values.

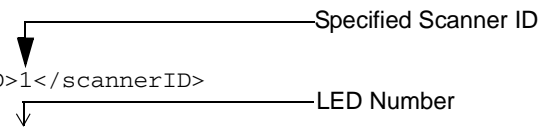
DEVICE_LED_ON

Description: Turn on a given LED* of a specified scanner

Asynchronous supported: N/A

Supported Scanner Communication Protocols: IBM Handheld, SNAPi

inXml:



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

outXml: null

* Refer to the Product Reference Guide of the scanner for supported LED values.

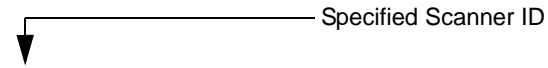
DEVICE_PULL_TRIGGER

Description: Pull the trigger of a specified scanner

Asynchronous supported: N/A

Supported Scanner Communication Protocols: SNAPi

inXml:



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

outXml: null

DEVICE_RELEASE_TRIGGER

Description: Release the pulled trigger of a specified scanner


Asynchronous supported: N/A

Supported Scanner Communication Protocols: SNAPI

inXml:

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

Specified Scanner ID



outXml: null

SCAN_DISABLE

Description: Disable scanning on a specified scanner

Asynchronous supported: N/A

Supported Scanner Communication Protocols: SNAPI, IBM Handheld, Nixdorf Mode B

inXml:

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

Specified Scanner ID



outXml: null

SCAN_ENABLE

Description: Enable scanning on a specified scanner

Asynchronous supported: N/A

Supported Scanner Communication Protocols: SNAPI, IBM Handheld, Nixdorf Mode B

inXml:

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

Specified Scanner ID



outXml: null

SET_PARAMETER_DEFAULTS

Description: Set parameters to default values of a specified scanner

Asynchronous supported: No

Supported Scanner Communication Protocols: SNAPI

inXml:

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

Specified Scanner ID



outXml: null

DEVICE_SET_PARAMETERS

Description: Set parameter(s) of a specified scanner temporarily. Parameters set using this command are lost after the next power down.

Asynchronous supported: No

Supported Scanner Communication Protocols: SNAPi

inXml: Specified Scanner ID

```
<inArgs>
  <scannerID>1</scannerID>
  <cmdArgs>
    <arg-xml>
      <attrib_list>
        <attribute>
          <id>145</id>
          <datatype>B</datatype>
          <value>0</value>
        </attribute>
      </attrib_list>
    </arg-xml>
  </cmdArgs>
</inArgs>
```

outXml: null

✓ **NOTE** Refer to the Attribute Data Dictionary (p/n 72E-149786-xx) for attribute numbers, types and possible values.

SET_PARAMETER_PERSISTANCE

Description: Set parameter(s) of a specified scanner persistently. Parameters set using this command are persistent over power down and power up cycles.

Asynchronous supported: No

Supported Scanner Communication Protocols: SNAPi

inXml: Specified Scanner ID

```
<inArgs>
  <scannerID>1</scannerID>
  <cmdArgs>
    <arg-xml>
      <attrib_list>
        <attribute>
          <id>145</id>
          <datatype>B</datatype>
          <value>0</value>
        </attribute>
      </attrib_list>
    </arg-xml>
  </cmdArgs>
</inArgs>
```

outXml: null

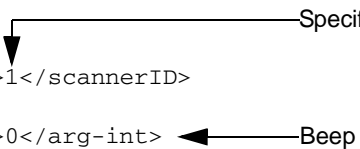
DEVICE_BEEP_CONTROL

Description: Beep the beeper of a specified scanner

Asynchronous supported: N/A

Supported Scanner Communication Protocols: IBM Handheld, SNAPi

inXml:



```
<inArgs>
  <scannerID>1</scannerID>
  <cmdArgs>
    <arg-int>0</arg-int>
  </cmdArgs>
</inArgs>
```

[Table 3-11](#) lists the Beep IDs for all XML examples above.

Table 3-11 *Beep IDs for Scanners*

Beep ID	Beep Name
0	ONE SHORT HIGH
1	TWO SHORT HIGH
2	THREE SHORT HIGH
3	FOUR SHORT HIGH
4	FIVE SHORT HIGH
5	ONE SHORT LOW
6	TWO SHORT LOW
7	THREE SHORT LOW
8	FOUR SHORT LOW
9	FIVE SHORT LOW
10	ONE LONG HIGH
11	TWO LONG HIGH
12	THREE LONG HIGH
13	FOUR LONG HIGH
14	FIVE LONG HIGH
15	ONE LONG LOW
16	TWO LONG LOW
17	THREE LONG LOW
18	FOUR LONG LOW
19	FIVE LONG LOW
20	FAST HIGH LOW HIGH LOW

Table 3-11 Beep IDs for Scanners (Continued)

Beep ID	Beep Name
21	SLOW HIGH LOW HIGH LOW
22	HIGH LOW
23	LOW HIGH
24	HIGH LOW HIGH
25	LOW HIGH LOW

outXml: null

REBOOT_SCANNER

Description: Reboot a specified scanner

Asynchronous supported: N/A

Supported Scanner Communication Protocols: IBM Handheld, SNAPi

inXml: Specified Scanner ID

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

outXml: null

DEVICE_CAPTURE_IMAGE

Description: Change a specified scanner to snapshot mode. While in this mode, an imaging scanner blinks the green LED at one second intervals to indicate it is not in standard operating (decode) mode. The scanner comes to its standard operating mode after a trigger pull or the snapshot time out is exceeded. After a trigger pull, the CoreScanner driver triggers an ImageEvent containing the captured image (see [ImageEvent on page 3-5](#)).

Asynchronous supported: N/A

Supported Scanner Communication Protocols: SNAPi

inXml: Specified Scanner ID

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

outXml: null

DEVICE_CAPTURE_VIDEO

Description:

Change a specified scanner to video mode. In this mode, the imaging scanner behaves as a video camera as long as the trigger is pulled. When the trigger is released, the scanner returns to Decode Mode. As long as the trigger is pulled, the CoreScanner driver triggers VideoEvents that contain the video data (see [VideoEvent on page 3-6](#)).

Asynchronous supported:

N/A

Supported Scanner Communication Protocols: SNAPi

inXml:

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

Specified Scanner ID

outXml:

null

ATTR_GETALL

Description:

Get all the attributes of a specified scanner. A synchronous call of this method returns an outXML like the example below. An asynchronous call of this event triggers a CommandResponseEvent (Refer to CommandResponseEvent on page 3-42).

Asynchronous supported:

Yes

Supported Scanner Communication Protocols: IBM Handheld, SNAPi

inXml:

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

Specified Scanner ID

outXml:

```
<?xml version="1.0" encoding="UTF-8" ?>
<outArgs>
```

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

Scanner ID of Data
Receiving

```
  <arg-xml>
```

```
    <modelnumber>DS670-SR20001ZZR</modelnumber>
```

```
    <serialnumber>7116000501003</serialnumber>
```

```
    <GUID>A2E647DED2163545B18BCEBD0A2A133D</GUID>
```

Assert Tracking
Information of the Scanner

```
  <response>
```

```
    <opcode>5000</opcode>
```

Method Response
Received

```
    <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 Numbers

[illegible]

```

        <attribute name="">20013</attribute>
    </attrib_list>
</response>
</arg-xml>
</outArgs>

```



NOTE Refer to the Attribute Data Dictionary (p/n 72E-149786-xx) for attribute numbers, types and possible values.

ATTR_GET

Description:

Query the values of attribute(s) of a specified scanner. An synchronous call of this method returns outXML like the example below. An asynchronous call of this event triggers a [CommandResponseEvent](#) (see [CommandResponseEvent on page 3-11](#)).

Asynchronous supported:

Yes

Supported Scanner Communication Protocols: IBM Handheld, 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>

```

Scanner ID of Data Receiving

Assert Tracking Information of the Scanner

Method Response Received

Attribute Numbers

Attribute Data Type

Permissions of the Attribute

Attribute Value

```

        <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 0x00 0x00</value>
    </attribute>
</attrib_list>
</response>
</arg-xml>
</outArgs>

```

ATTR_GETNEXT

Description:

Query the value of the next attribute to a given attribute of a specified scanner. A synchronous call of this method returns an outXML like the example below. An asynchronous call of this event triggers a [CommandResponseEvent](#) (see [CommandResponseEvent on page 3-11](#)).

Asynchronous supported:

Yes

Supported Scanner Communication Protocols: IBM Handheld, 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>

```

Scanner ID of Data Receiving

Asset Tracking Information of the Scanner

Method Response Received

```

<attrib_list>
  <attribute>
    <id>15</id> ← Attribute Number
    <name></name>
    <datatype>F</datatype> ← Attribute Data Type
    <permission>RWP</permission> ← Permissions of the Attribute
    <value>True</value> ← Attribute Value
  </attribute>
</attrib_list>
</response>
</arg-xml>
</outArgs>

```



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

Description:

Set the values of attribute(s) of a specified scanner.
Attribute(s) set using this command are lost after the next power down.

Asynchronous supported:

N/A

Supported Scanner Communication Protocols: IBM Handheld, SNAPi

inXml:

```

<inArgs>
  ← Specified Scanner ID
  <scannerID>1</scannerID>
  <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>
</inArgs>

```

outXml:

null

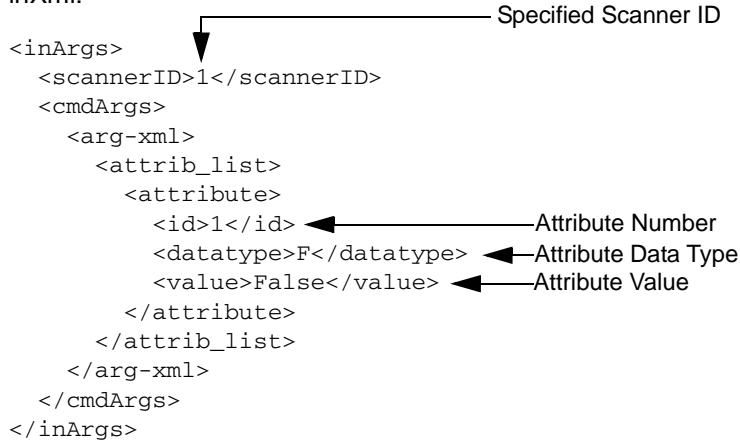
ATTR_STORE

Description: Store the values of attribute(s) of a specified scanner. Attribute(s) store using this command are persistent over power down and power up cycles.

Asynchronous supported: N/A

Supported Scanner Communication Protocols: IBM Handheld, SNAPi

inXml:



```
<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>
```

outXml: null

GET_DEVICE_TOPOLOGY

Description: Get the topology of devices that are connected to the calling system

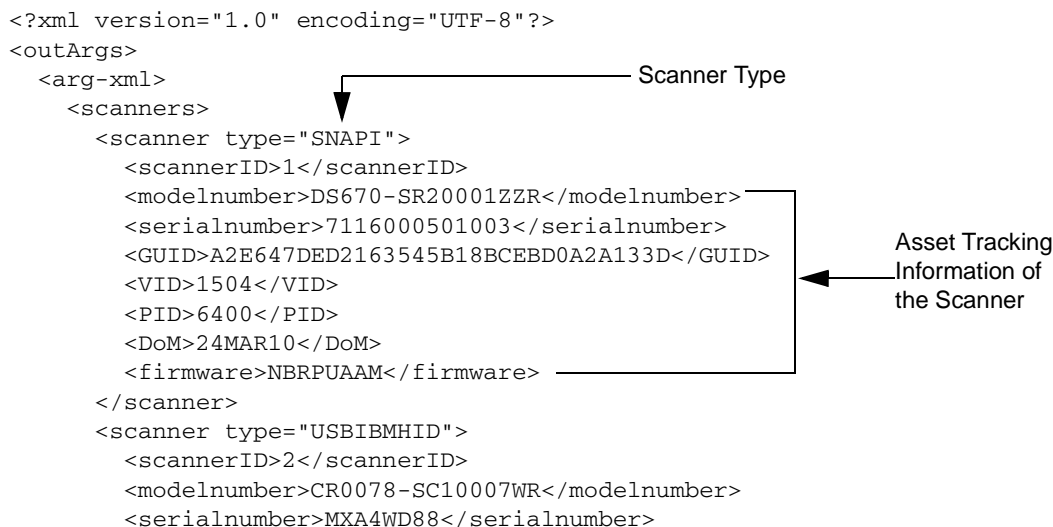
Asynchronous supported: No

Supported Scanner Communication Protocols: IBM Handheld, 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>
      </scanner>
    </scanners>
  </arg-xml>
</outArgs>
```

```

<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>PAAAJ500-002-N25</firmware>
</scanner>
</scanners>
</arg-xml>
</outArgs>

```

← Cascaded Scanner

START_NEW_FIRMWARE

Description: Start the updated firmware. This will cause a reboot of the specified scanner.

Asynchronous supported: N/A

Supported Scanner Communication Protocols: IBM Handheld, SNAPi

inXml:

← Specified Scanner ID

```

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

```

outXml: null

UPDATE_FIRMWARE

Description: Update the firmware of the specified scanner. A user can specify the bulk firmware update option for faster firmware download in SNAPi mode. If an application registered for ScanRMDEvents, it will receive ScanRMDEvents as described [on page 3-11](#).

Asynchronous supported: N/A

Supported Scanner Communication Protocols: IBM Handheld, SNAPi

inXml:

← Specified Scanner ID

```

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

```

← Path to the DAT File

← Bulk Update Option

outXml: null

DEVICE_SET_SERIAL_PORT_SETTINGS

Description: Set the serial port settings of a NIXDORF Mode B Scanner

Asynchronous supported: N/A

Supported Scanner Communication Protocols: a NIXDORF Mode B

inXml:

```

<inArgs>
  <scannerID>1</scannerID>
  <cmdArgs>
    <arg-int>5</arg-int>
    <arg-int>9600,8,0,0,1</arg-int>
  </cmdArgs>
</inArgs>

```

outXml: null

Diagram labels for inXml:

- Specified Scanner ID (points to <scannerID>1</scannerID>)
- Number of Parameters (points to <arg-int>5</arg-int>)
- Serial Port Settings (points to <arg-int>9600,8,0,0,1</arg-int>)

DEVICE_SWITCH_HOST_MODE

Description: Switch the USB host mode of a specified scanner. This operation causes a reboot of the device. When the specified scanner is in HID Keyboard mode, the only allowed target host variants are IBM Handheld and SNAPI. A user can configure the switching host mode as a silent switch (without the typical device reboot beeps) and keep the targeted host mode as the permanent host mode of the device by changing parameters of inXML.

Asynchronous supported: N/A

Supported Scanner Communication Protocols: IBM Handheld, SNAPI, HID Keyboard

inXml:

```

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

```

Diagram labels for inXml:

- Specified Scanner ID (points to <scannerID>1</scannerID>)
- String Code for Target Host Variant (points to <arg-string>XUA-45001-1</arg-string>)
- Silent Switch Option (points to <arg-bool>TRUE</arg-bool>)
- Permanent Change Option (points to <arg-bool>FALSE</arg-bool>)

[Table 3-12](#) lists the string codes for USB host variants.

Table 3-12 USB Host Variants

Host Variant	String Code
USB-IBMHID	XUA-45001-1
USB-IBMTT	XUA-45001-2
USB-HIDKB	XUA-45001-3
USB-OPOS	XUA-45001-8
USB-SNAPI with Imaging	XUA-45001-9
USB-SNAPI without Imaging	XUA-45001-10

outXml: null

KEYBOARD_EMULATOR_ENABLE

Description: This setting enables/disables the keyboard emulation mode of the connected scanners which are in IBM Handheld, SNAPI and NIXDORF Mode B host modes.

Asynchronous supported: No

Supported Scanner Communication Protocols: N/A

inXml:

```
<inArgs>
  <cmdArgs>
    <arg-bool>TRUE</arg-bool> ← Keyboard Emulator State
  </cmdArgs>
</inArgs>
```

outXml: null

✓ **NOTE** Any HIDKB Emulator-related settings that are changed using the API are temporary. These settings revert to the values in the config.xml file after a restart of the CoreScanner service or a system reboot.

KEYBOARD_EMULATOR_SET_LOCALE

Description: Change the locale of the emulated keyboard

Asynchronous supported: No

Supported Scanner Communication Protocols: N/A

inXml:

```
<inArgs>
  <cmdArgs>
    <arg-int>1</arg-int> ← Keyboard Emulator Language Locale ID
  </cmdArgs>
</inArgs>
```

[Table 3-13](#) lists the Language Locale ID for the XML code above.

Table 3-13 *Language Locale IDs*

Locale	Value
English	0
French	1

outXml: null

KEYBOARD_EMULATOR_GET_CONFIG

Description: Gets the current configuration of the HID Keyboard Emulator from the config.xml file.

Asynchronous supported: No

Supported Scanner Communication Protocols: N/A

inXml:

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

outXml:

```
<outArgs>
  <arg-xml>
    <KeyEnumState>1</KeyEnumState> ← Keyboard Emulator State
    <KeyEnumLocale>0</KeyEnumLocale> ← Keyboard Emulator Language Locale ID
  </arg-xml>
</outArgs>
```

Error/Status Codes

Table 3-14 *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_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
ERROR_XMLREADER_NOT_CREATED	301	XML Reader could not be instantiated
ERROR_XMLREADER_INPUT_NOT_SET	302	Input for XML Reader could not be set
ERROR_XMLREADER_PROPERTY_NOT_SET	303	XML Reader property could not be set
ERROR_XMLWRITER_NOT_CREATED	304	XML Writer could not be instantiated

Table 3-14 *Error and Status Codes (Continued)*

Error/ Status Code	Value	Description
ERROR_XMLWRITER_OUTPUT_NOT_SET	305	Output for XML Writer could not be set
ERROR_XMLWRITER_PROPERTY_NOT_SET	306	XML Writer property could not be set
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 exceed length
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	FW Update failed in scanner
ERROR_FW_READ_FAILED_DATFILE	502	Failed to read DAT file
ERROR_FW_UPDATE_INPROGRESS	503	Firmware Update is in progress (cannot proceed another FW Update or another command)
ERROR_FW_UPDATE_ALREADY_ABORTED	504	Firmware update is already aborted
ERROR_FW_UPDATE_ABORTED	505	FW Update aborted
ERROR_FW_SCN_DETACHED	506	Scanner is disconnected while updating firmware
STATUS_FW_SWCOMP_RESIDENT	600	The software component is already resident in the scanner

CHAPTER 4 TEST UTILITIES & SOURCE CODE

Overview

This chapter provides information about testing and evaluation of the Motorola Scanner SDK's software components using the test utilities provided in the SDK.

For a list of the most commonly requested topics within this guide, see [Quick Startup](#) in the back of the guide.



NOTE For a complete list of attribute (parameter) numbers and their definitions, download the Attribute Data Dictionary (p/n 72E-149786-xx) from <http://MotorolaSolutions.com/WindowsSDK>. Attributes include configuration parameters, monitored data and asset tracking information.

Test Utilities Provided in the SDK

The Motorola Scanner SDK includes the following test utilities:

- Motorola Scanner SDK C++ Sample Application
- Motorola Scanner SDK C# .Net Sample Application

Each test utility demonstrates the main functionalities of the SDK. You can gain an understanding of the Motorola Scanner SDK using these test utilities. This section also describes how to use the test utilities' functionality.

✓ **NOTE** You may need to install the Microsoft®.Net Framework v2.0 or later to execute C# .Net Sample application. If so, Microsoft detects and informs the user of this requirement.

The Motorola Scanner SDK Test Utilities support the following functionality:

- Discovery of asset tracking information
- Scan a bar code
- Capture Image and Video
- Attribute query and setting
- Host Variant switching
- Firmware upgrade.

Motorola Scanner SDK C++ Sample Application

The Motorola Scanner SDK C++ Sample Application enables you to simulate an application that communicates with the Motorola Scanner SDK. The utility demonstrates the functionality of the SDK. It includes C++ source code and its solution and project files for further reference.

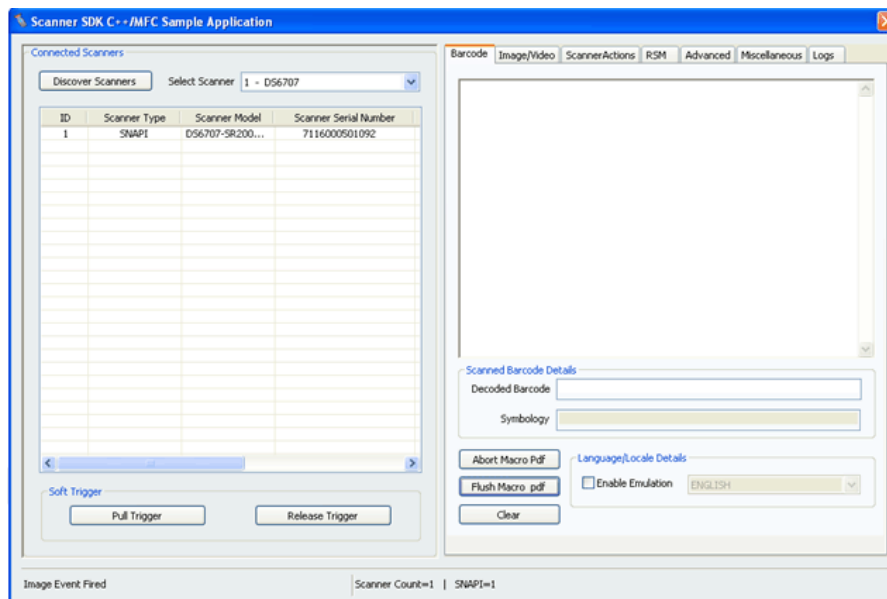


Figure 4-1 C++ Sample Application

Motorola Scanner SDK C#.Net Sample Application

The Motorola Scanner SDK C#.Net Sample Application enables you to simulate an application that communicates with the Motorola Scanner SDK. The utility demonstrates the functionality of the SDK. It includes C#.Net source code and its solution and project files for further reference.

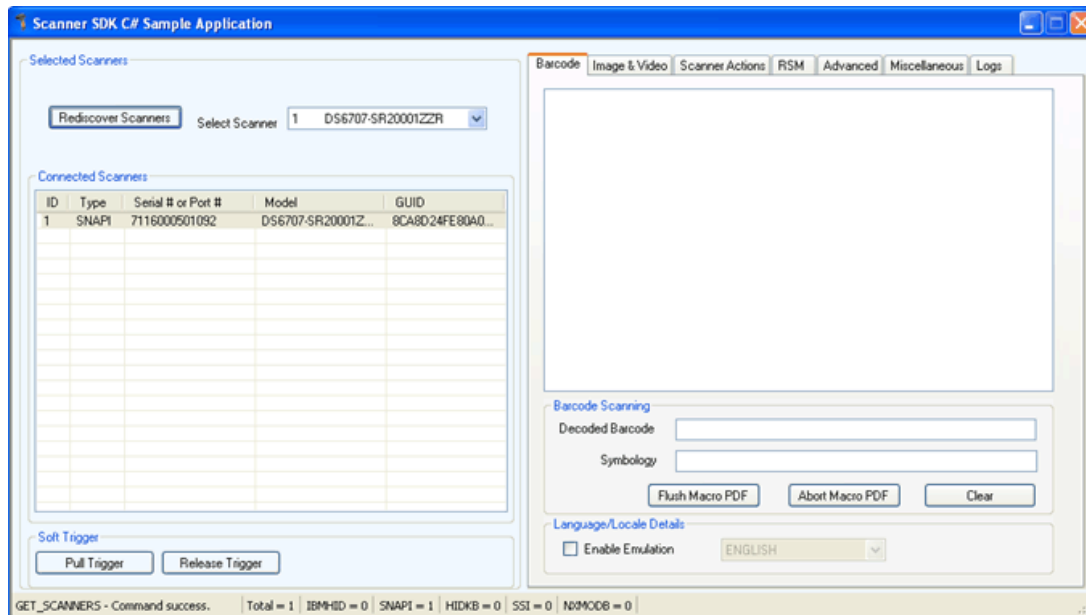


Figure 4-2 C# Sample Application

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

Button or Field	Description
Discover Scanners	Invokes Open, GetScanners methods and register for all the events.
Select Scanner	Select the scanner you want to invoke the command
Connected Scanners	List all the connected scanners regardless of the mode
Pull Trigger	Soft Pull Trigger the scanner for Bar code, Image and Video actions
Release Trigger	Soft Release Trigger the scanner for Bar code, Image and Video actions
Bar Code Tab	
Flush Macro PDF	Flush Macro PDF bar code buffer
Abort Macro PDF	Abort Macro PDF continues read
Clear	Clear the Bar code data area
Decoded Bar Code	Display label value of the scanned bar code
Symbology	Display the symbology of scanned bar code
Enable Emulation	Enable Simulated HID Keyboard Output
Image/Video Tab	
Image	Invoke image capture mode

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

Button or Field	Description
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 peeper of the scanner.
Reboot Scanner	Reboot the scanner.
LED	Light the LED(s) on the scanner.
Switch Host Variant	Switch the scanner host type from current type to desired type, user will have the option to select silent feature and variant change persistent and non-persistent .
RSM Tab	
Get All IDs	Get all supported attribute IDs from the selected scanner.
Get Value	Select one or more attribute IDs and get the value for them.
Next Value	Get the next attributes value given the current attribute number.
Store Value	Store value(s) for selected attribute(s).
Set Value	Set value(s) for selected attribute(s).
Select All	Select all the attribute IDs at the RSM data viewer.
Clear All	Clear all the attribute data at the RSM data viewer.
Clear All Values	Clear all the 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	Updated firmware and launch the new firmware on the scanner.
Browse	Browse the Firmware file (*.DAT).
Update	Initiate firmware update process.
Abort	If you want to abort firmware update process.
Launch	Once firmware update finishes launch the new firmware in the scanner.
Claim Scanner	Exclusively claim and declaim the scanner for this application.
Miscellaneous Tab	
SDK Version	Get the scanner SDK version.

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

Button or Field	Description
Get Device Topology	Get the scanner device topology, this is useful to get an idea of scanner topology for cascaded scanners.
Serial Interface Settings	Serial interface settings for serial scanners.
Logs Tab	
Event Log	Command and event log, logs commands initiated.
XML log	Displays Output of each function if an output exists.
Clear Event Log	Clear command and event log area.
Clear XML Log	Clear XML log area.

How to Verify Scanner SDK Functionality

This section guides you through a series of use cases and test cases of the Motorola Scanner SDK and its functionality.

See [Basic Installation Verification on page 2-12](#) for more information.

Scanner Discovery / Asset Tracking Information / Validating Successful SDK Installation

1. Connect a Motorola USB scanner(s) to the computer and put the scanner into USB OPOS (Hand Held) or USB SNAPI mode by scanning one of the bar codes below.



USB (IBM Hand Held)



USB SNAPI

2. Launch the Motorola Scanner SDK Sample Utility by selecting *Start > All Programs > Motorola Scanner > Scanner SDK > Scanner SDK Sample Application (C++)* or *Scanner SDK Sample Application (C#.Net)*.

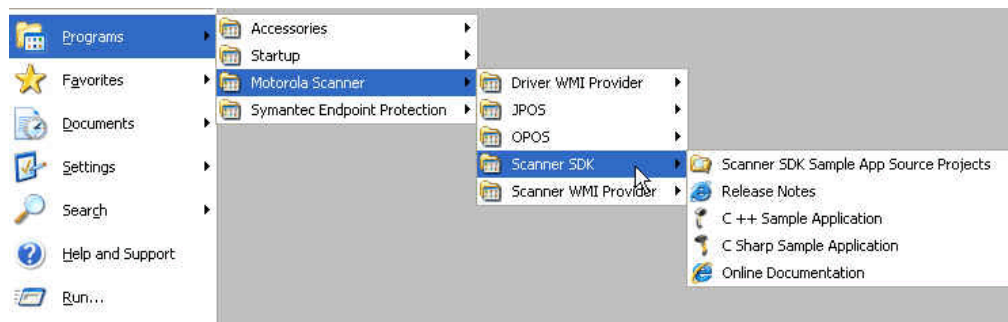


Figure 4-3 Start Scanner SDK Sample Application (C++) or C#.Net

3. Click **Discover Scanners** to display all the connected scanners in the *Connected Scanners* area.
4. Clicking **Discover Scanners** in the sample application 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 is filled with the number of events you want to register. In the example above, number of event it wants to register is "6". The second <arg-int> tag is filled with the event ids that you want to register separated by the commas (","). See event IDs in [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 how to call [Open](#), [ExecCommand](#) and [GetScanners](#) APIs.

1. The GetScanners API call produces XML code as follows:

```
<?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 Data Representation of the GetScanners Output in this Example

Scanner Information	Value	Description
Scanner ID	1	A unique ID assigned for a scanner from the SDK; any scanner specific method execute from ExecCommand should point to a scanner ID
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.

All discovered scanners are presented in the *Connected Scanners* window (*Figure 4-4*) by processing the XML received from the *GetScanners* command along with their asset tracking information returned by querying device parameters. The detection of scanners indicates the SDK was installed successfully.

Click **Discover Scanners** to display the connected scanners.

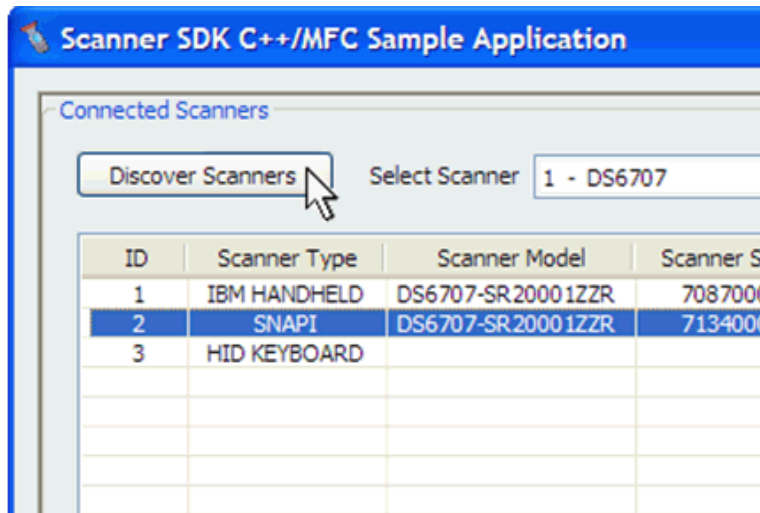


Figure 4-4 Connected Scanners

Bar Code Scanning

- Connect and discover a scanner (see *Scanner Discovery / Asset Tracking Information / Validating Successful SDK Installation on page 4-6*).
- Scan a bar code and its decoded data is returned in the form of XML data and displayed on the *Barcode* tab. To illustrate the typical implementation, the sample application will also display only the "Bar code" data below the XML data.

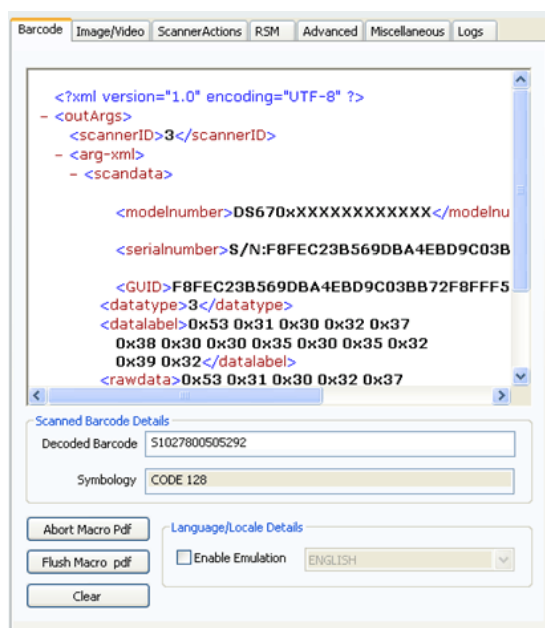


Figure 4-5 Decoded Bar Code Data

Example

1. Scan the following sample bar code after discovering the scanner in the sample application (see [Scanner Discovery / Asset Tracking Information / Validating Successful SDK Installation on page 4-6](#)).



Sample Bar Code

2. The following XML is returned:

```
<?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>
```

3. By processing the XML above, the sample application displays the decoded bar code in the *Decoded Bar code* text box and the symbology in the *Symbology* text box.

Language/Locale Details

1. Toggle the *Enable Emulation* check box to enable/disable *Simulated HID Keyboard Output*.
2. Select the language locale from the drop down menu.

The sample application first retrieves the current config.xml file (see [Simulated HID Keyboard Output on page 2-11](#) by executing an *ExecCommand* API call with the *KEYBOARD_EMULATOR_GET_CONFIG* method and an empty inXML. It receives outXML as shown below:

```
inXML:
<inArgs></inArgs>

outXML:
<outArgs>
  <arg-xml>
    <KeyEnumState>1</KeyEnumState>
    <KeyEnumLocale>0</KeyEnumLocale>
  </arg-xml>
</outArgs>
```

The sample application processes the XML above and populates the user interface. The <KeyEnumState> tag indicates the current state of Simulated HID Keyboard Output, where enabled = 1 and disabled = 0. The <KeyEnumLocale> tag indicates the language locale number currently active with the CoreScanner service. The value of "0" above indicates English.

Use the [ExecCommand](#) API call with the [KEYBOARD_EMULATOR_ENABLE](#) method and following inXML to enable/disable Simulated HID Keyboard Output.

```
<inArgs>
  <cmdArgs>
    <arg-bool>TRUE</arg-bool>
  </cmdArgs>
</inArgs>
```

To enable HID KB Emulator use "TRUE" in <arg-bool> tags and "FALSE" to disable it.

Use the [ExecCommand](#) API call with the [KEYBOARD_EMULATOR_SET_LOCALE](#) method and following inXML to change the language locale.

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

Set the <KeyEnumLocale> tag value to "1" for French and "0" for English.

Capture Image and Video

1. Connect and discover an imaging scanner (see [Scanner Discovery / Asset Tracking Information / Validating Successful SDK Installation on page 4-6](#)).
2. Select a "SNAPI" mode scanner ID from the *Select Scanner* drop-down box. Your selection is then reflected in the *Connected Scanners* window.

✓ **NOTE** If no SNAPI scanner is shown in the *Connected Scanners* window, you must connect an imaging scanner that supports image/video transfer. For a list of scanners supporting this functionality, see [Table 2-2 on page 2-3](#):

Alternatively, select "SNAPI" mode scanner in the *Connected Scanners* area. Your selected Scanner's ID is displayed in the *Select Scanner* drop-down combo box.

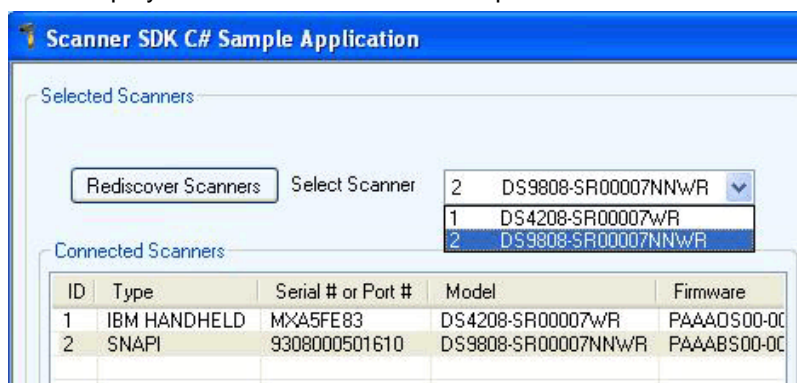


Figure 4-6 Scanner Selection

3. Go to the *Image & Video* tab.
4. Select an image type of JPG, TIFF or BMP.
5. Selecting the image type in the sample application executes an [ExecCommand](#) API call using the [DEVICE_SET_PARAMETERS](#) 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 is filled with the scanner's ID selected in the *Connected Scanners* list of the sample application. The <id> tag contains the image file type parameter of the selected scanner. In the XML example above, this value is 304. The value 4 indicates the image type the user should get from the scanner. See [Table 4-4](#) for valid Image Types.

Table 4-4 *Image Types*

Image Type	Value
BMP_FILE_SELECTION	3
TIFF_FILE_SELECTION	4
JPEG_FILE_SELECTION	1

- ✓ **NOTE** These values may change with the scanner model. Refer to the scanner Product Reference Guide for more information on scanner parameters. For more information about parameter settings, see [Parameter Setting \(Device Configuration\) on page 4-19](#).

6. Check *Enable Video View Finder* and click either **Image** to put the scanner into image capture mode or **Video** to put the scanner into video capture mode.
7. Checking *Enable Video View Finder* in the sample application will execute an [ExecCommand](#) API call with the [DEVICE_SET_PARAMETERS](#) method and 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 selected scanner's ID from the *Connected Scanners* list of the sample application. 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.

8. Click **Image** in the sample application. **Image** executes an [ExecCommand](#) API call using the [DEVICE_CAPTURE_IMAGE](#) method with the XML code below. Click **Video** to execute an [ExecCommand](#) API call using the [DEVICE_CAPTURE_VIDEO](#) method with the following XML code.

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

9. Click **Pull Trigger** on the bottom left side of the utility to capture an image. If the scanner was placed into video capture mode in the previous step, click **Pull Trigger** once to start video capture and click **Release Trigger** to stop video capture.

10. Clicking **Pull Trigger** or **Release Trigger** in the sample application 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.

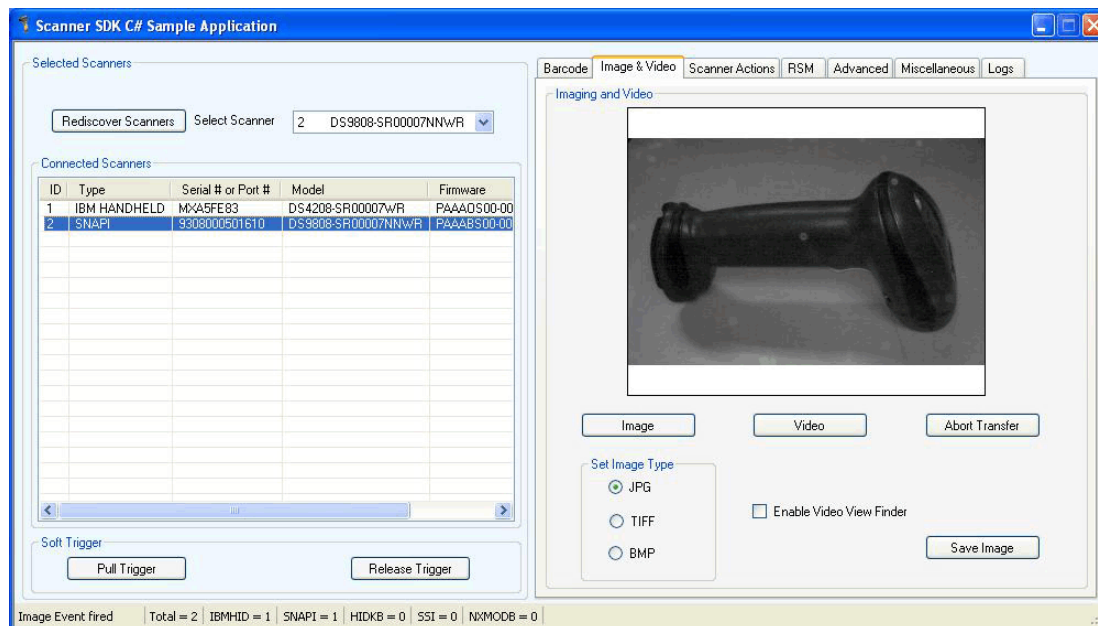


Figure 4-7 Captured Image Displayed on the Image & Video Tab

11. If you registered with ImageEvent (see [Register for COM Events on page 5-2](#)) you receive an image event for the performed pull trigger when in image mode.
12. If you registered with VideoEvent (see [Register for COM Events on page 5-2](#)) you receive a video event for the performed pull trigger when in video mode.

Beep the Beeper

Motorola Scanners are capable of sounding the beeper by invoking the Beeper method from the host system.

1. Connect and discover a scanner (see [Scanner Discovery / Asset Tracking Information / Validating Successful SDK Installation on page 4-6](#)).
2. Select a "SNAPI" or "OPOS/IBM OPOS" mode scanner ID from the *Select Scanner* drop-down box. Your selection is reflected in the *Connected Scanners* window (see [Figure 4-6 on page 4-10](#)).
3. Select the desired beep sequence from the list defined on the *Scanner Actions* tab as shown in [Figure 4-8](#).
4. Click **Beep**.

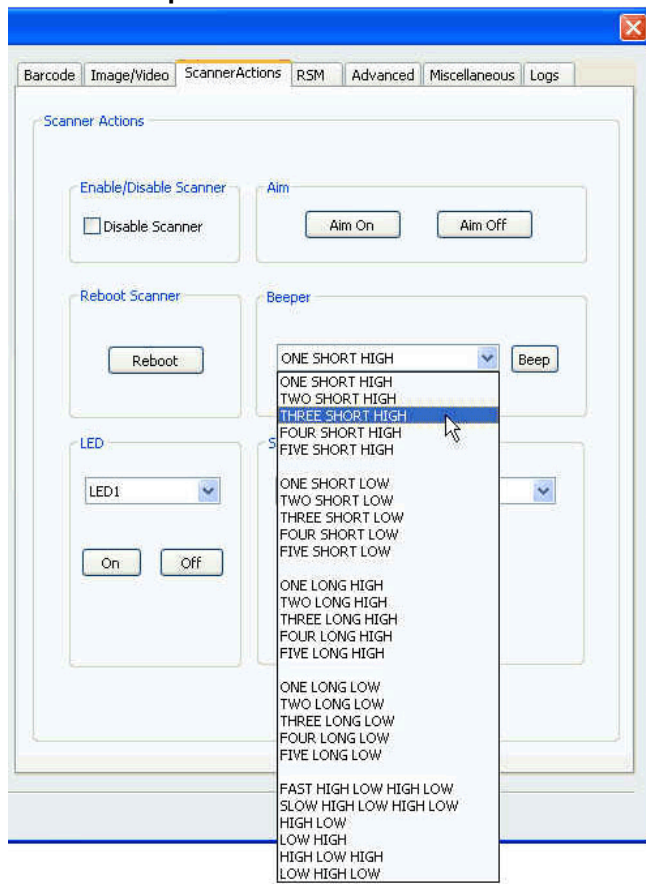


Figure 4-8 Beep Values

5. Clicking **Beep** in the sample application executes an *ExecCommand* API call with the *DEVICE_BEEP_CONTROL* method and following XML code:

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

✓ **NOTE** The <scannerID> tag in the XML is filled with the scanner's ID selected in the *Connected Scanners* list of the sample application. The <arg-int> tag in the XML is filled with the beep's ID selected in the *Beeper* drop-down list shown in [Figure 4-10 on page 4-17](#). See [Table 3-11 on page 3-20](#) for beep IDs.

6. You can sound any of the beeps in the table [Beep IDs for Scanners on page 3-20](#) by changing the value of the <arg-int> tag in the XML code. Successful execution of the command will return the status parameter as "0".

Flash the LED

Motorola scanners are capable of flashing an LED by initiating the flash LED method from the host system.

1. Connect and discover a scanner (see [Scanner Discovery / Asset Tracking Information / Validating Successful SDK Installation on page 4-6](#)).
2. Select a "SNAPI" or "OPOS/IBM OPOS" mode scanner ID from the *Select Scanner* drop-down box. Your selection is reflected in the *Connected Scanners* window (see [Figure 4-6 on page 4-10](#)).
3. Select the desired LED from the list, defined on the *Scanner Actions* tab (see [Figure 4-9 on page 4-14](#)).
4. Click **On** to light the LED and **Off** to turn it off.

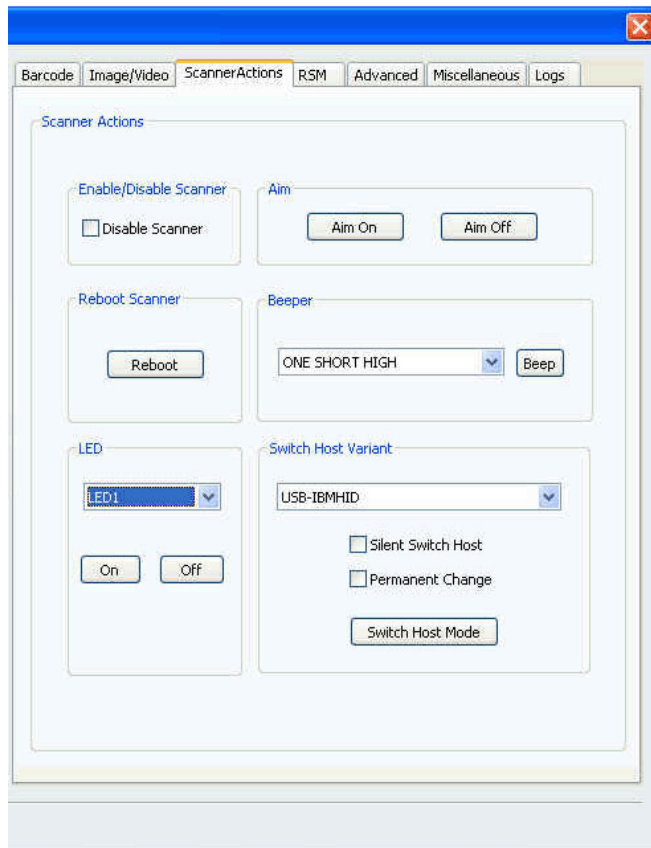


Figure 4-9 LED Selection

5. Clicking **On** in the sample application executes an *ExecCommand* API call with the *DEVICE_LED_ON* method and the following XML code:

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

✓ **NOTE** The `<scannerID>` tag in the XML is filled with the scanner's ID selected in the *Connected Scanners* list of the sample application. The `<arg-int>` tag in the XML is filled with the LED number selected in the *LED* drop-down list shown in [Figure 4-9](#). Refer to the scanner's Product Reference Guide for its supported LED values.

6. You can light any LED supported by the scanner by changing the value of the `<arg-int>` tag value.

- Clicking **Off** in the sample application executes 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. The only difference between these commands is 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. Inside `<cmdArgs>`, there can be `<arg-string>`, `<arg-bool>` and `<arg-int>` tags. You can execute different commands for the same XML by changing the method parameter in *ExecCommand*.

Querying Attributes and Parameters

To query parameters from a specific device, such as the *Date of Manufacture* and *Firmware Version*, use the following procedure.

- Connect and discover a scanner (see *Scanner Discovery / Asset Tracking Information / Validating Successful SDK Installation on page 4-6*).
- Select the scanner you want to query from the list of *Connected Scanners* and then select the *RSM* tab.
- Click **Get All IDs** to retrieve the entire list of supported attribute IDs of the selected scanner. This operation 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 scanner's ID selected in the *Connected Scanners* list of the sample application.

- The sample application receives the XML output below and displays the corresponding attribute IDs on the grid (see *Figure 4-10 on page 4-17*).

```
<?xml version="1.0" encoding="UTF-8" ?>
<outArgs>
  <scannerID>1</scannerID>
  <arg-xml>
    <modelnumber>DS670-SR20001ZZR</modelnumber>
    <serialnumber>7116000501003</serialnumber>
    <GUID>A2E647DED2163545B18BCEBD0A2A133D</GUID>
  </arg-xml>
  <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 Attribute Data Dictionary (p/n 72E-149786-xx).

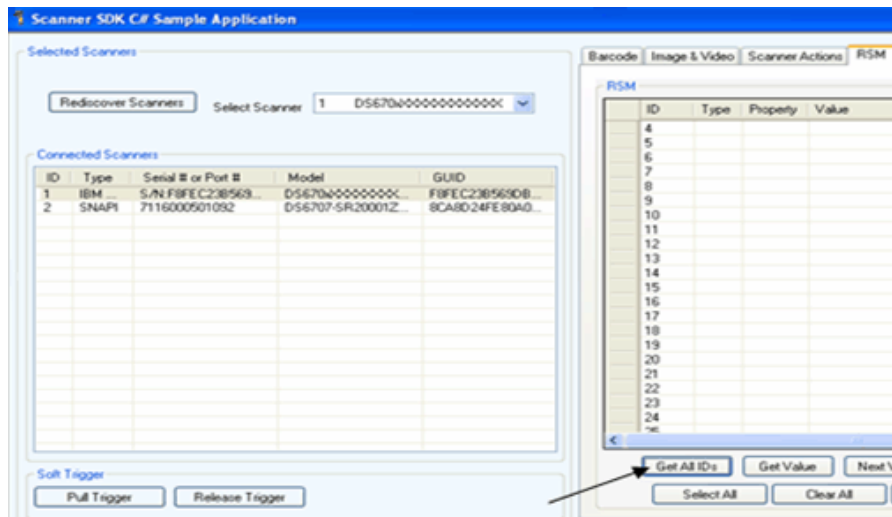


Figure 4-10 Get RSM IDs

- To query attributes, select attribute IDs and click "Get Value" to view the attribute values. This operation 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 scanner's ID selected in the *Connected Scanners* list and the <attrib_list> tag with the attribute IDs selected in the RSM grid.

For example, if you want to retrieve the values of the *Date of Manufacture*, *Firmware Version*, *UPC -A status*, *Beeper Volume* and *ADF Rule* parameters, you need to know their attribute IDs. [Table 4-5](#) shows the corresponding IDs. For detailed information, refer to the Attribute Data Dictionary, p/n 72E-149786-xx. Selecting these attribute IDs in the grid of the sample application and clicking **Get Value** executes an *ExecCommand* API call with the *ATTR_GET* method and the XML shown above.

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

The sample application's RSM grid displays the output as in Figure 4-12 by processing the XML above.

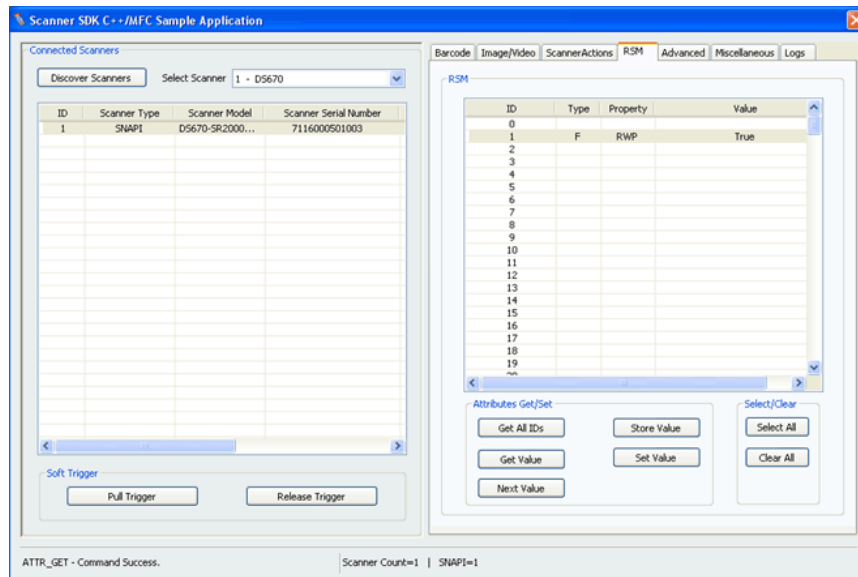


Figure 4-11 RSM Attribute Values for Selected IDs

Parameter Setting (Device Configuration)

To set parameters of a specific device, such as UPC-A status or Beeper Volume, use the following procedure.

1. Query the parameter (see [Querying Attributes and Parameters on page 4-15](#)).
2. To set an attribute, select and edit the attribute value in the *RSM* window data grid. Then select the entire row of the changed attribute and click **Set Value** or **Store Value**. Clicking these buttons execute an [ExecCommand](#) API call using the [ATTR_SET](#) or [ATTR_STORE](#) method and XML code shown below.

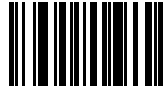
```
<inArgs>
  <scannerID>1</scannerID>
  <cmdArgs>
    <arg-xml>
      <attrib_list>
        <attribute>
          <id>1</id>
          <datatype>F</datatype>
          <value>False</value>
        </attribute>
      </attrib_list>
    </arg-xml>
  </cmdArgs>
</inArgs>
```

✓ **NOTE** The `<scannerID>` tag in the XML contains the scanner's ID selected from the *Connected Scanners* list and the `<attrib_list>` tag contains the `<attribute>` tags selected in the RSM grid.

Examples

These examples demonstrate how to enable/disable a symbology, program an ADF rule, control beeper volume and control LEDs.

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



Set All Defaults

Enable / Disable a Symbology

To disable the UPC-A symbology, determine the attribute ID of UPC-A by referencing the Attribute Data Dictionary (p/n 72E-149786-xx). The attribute ID of the UPC-A parameter is "1". To change and validate the setting, use the following procedure:

1. Put the scanner into USB OPOS (Hand Held) or USB SNAPI mode by scanning one of the bar codes in [Scanner Discovery / Asset Tracking Information / Validating Successful SDK Installation on page 4-6](#), or set the mode using the procedure in [Host Variant Switching on page 4-23](#).
2. Get the value of attribute ID 1. The value of this attribute should be "True" if you scanned the **Set All Defaults** bar code before beginning the example.
3. To disable the UPC-A attribute of a scanner, change the value of the attribute ID 1 to "False" in the *RSM* grid and click **Set Value** or **Store Value**.
4. The sample application then executes an [ExecCommand](#) API call with the [ATTR_SET](#) or [ATTR_STORE](#) method and the XML shown in [Parameter Setting \(Device Configuration\) on page 4-19](#).
5. If the command executed successfully, you can not scan the following UPC-A bar code.



Sample UPC-A Bar Code

Programming an ADF Rule

If you want to create an ADF rule to add the prefix "A" to any bar code and an "Enter" key after scanning a bar code, you must modify the ADF buffer of the scanner. According to the Attribute Data Dictionary (p/n 72E-149786-xx), the attribute ID of the ADF rule is 392.

To change and validate the setting:

USB Host Type = HID Keyboard Wedge

1. Scan the bar code below, or follow the procedure in [Host Variant Switching on page 4-23](#) 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 and scan the [Sample UPC-A Bar Code on page 4-20](#) while the text editor is the active window. The text "012345678912" is inserted into the editor window.
3. Put the scanner into USB OPOS (hand held) or USB SNAPI mode by scanning one of the bar codes in [Scanner Discovery / Asset Tracking Information / Validating Successful SDK Installation on page 4-6](#), or following the procedure in [Host Variant Switching on page 4-23](#) to switch the host mode.
4. In the sample application, change the value of the selected scanner's attribute 392 to:
0x01 0x0C 0x11 0xF4 0x14 0x10 0x47 0x0D.
5. Click **Store Value**.
6. The sample application then 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 and 2.
8. The text entered in Notepad is "A012345678912<Enter key>".

Beeper Volume Control

Suppose you want to change the beeper volume of the scanner. According to the Attribute Data Dictionary (p/n 72E-149786-xx), the corresponding 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 SNAP! mode by scanning one of the bar codes in [Scanner Discovery / Asset Tracking Information / Validating Successful SDK Installation on page 4-6](#), or following the procedure in [Host Variant Switching on page 4-23](#) to switch the host mode.
2. Scan the [Sample UPC-A Bar Code on page 4-20](#) and listen to the beeper carefully.
3. Select attribute ID 140 from the RSM attribute grid. Its value should be "0" (if the **Set All Defaults** bar code was scanned at the beginning of the example).
4. Change the value to "2" and click **Set Value** or **Store Value**.
5. The sample application then 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>
```

6. After successfully executing the command, scan the Sample UPC-A bar code again and note that the beeper volume is lower.



NOTE Changes made using the *Store Value* commands are permanent (persistent over power down and power up cycles). Changes made using the *Set Value* command are temporary (parameters set using this temporary command are lost after the next power down).

Beeper and LED Control

Suppose you want to beep the scanner or light the LED of the scanner. According to the Attribute Data Dictionary (p/n 72E-149786-xx), the corresponding attribute ID is 6000.

To change and validate this setting:

1. Put the scanner into USB OPOS (hand held) or USB SNAP! mode by scanning one of the bar codes in [Scanner Discovery / Asset Tracking Information / Validating Successful SDK Installation on page 4-6](#), or following the procedure in [Host Variant Switching on page 4-23](#) to switch the host mode.
2. To light the LED of the scanner 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 the XML above. 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. Refer to the Attribute Data Dictionary (p/n 72E-149786-xx) for Action Attribute values.

Host Variant Switching

1. Connect and discover a scanner (see [Scanner Discovery / Asset Tracking Information / Validating Successful SDK Installation on page 4-6](#)).
2. Under the *Scanner Action* tab, select a *Target Mode* from the drop-down menu in the *Switch Host Variant* area.
3. *Permanent Change* or *Silent Reboot* options (hidden by the *Target Mode* drop-down list) may be selected if desired.
4. Click **Switch Host Mode** and the scanner reboots and sets to the selected target mode.

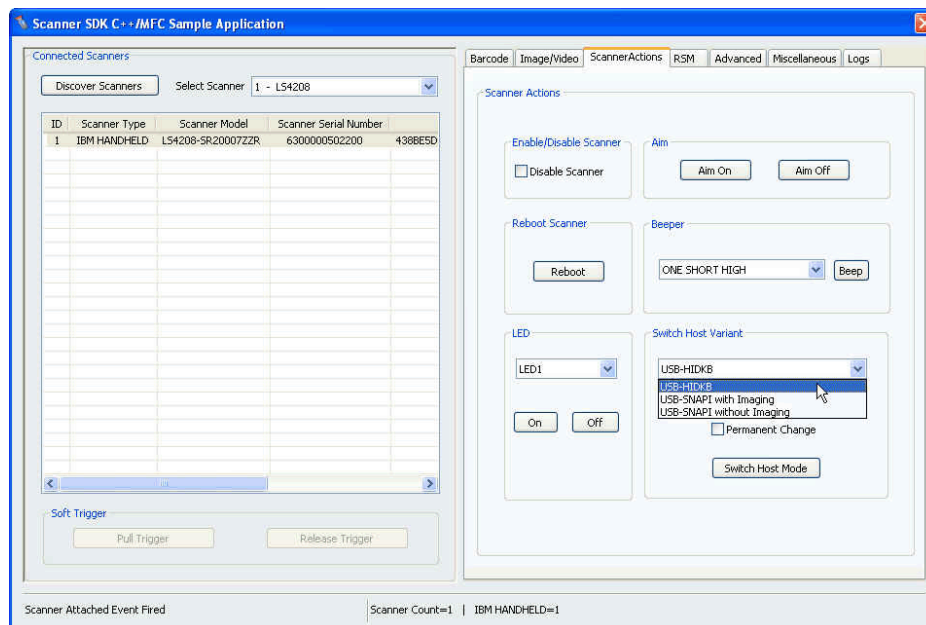


Figure 4-12 Changing Host Mode

5. Clicking **Switch Host Mode** in the sample application 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 above contains the scanner's ID selected from the *Connected Scanners* list of the sample application. The <arg-string> tag contains the string code of the target host variant selected from the drop-down list. These codes are referenced in [Table 3-12 on page 3-29](#). The first <arg-bool> tag contains the boolean value for the silent reboot option. A value of TRUE causes the scanner to reboot silently (without the typical reboot beeps). The second <arg-bool> tag contains the boolean value for the permanent change option.

A value of TRUE for this second <arg-bool> tag causes the target host variant to be persistent over power down and power up cycles. Otherwise the host variant change is temporary until the next reboot occurs.

When you are in HID Keyboard mode the only allowed target host variants are IBM Handheld USB and SNAPi.

See [Table 3-12 on page 3-29](#) for a list of string codes for USB host variants.

Firmware Upgrade

Firmware Upgrade Scenarios

Three firmware upgrade scenarios that should be considered are discussed below.

Scenario A: Loading a compatible, different version of firmware from the firmware already on the scanner

- Upgrading the firmware on a scanner includes two steps:
 1. The firmware file downloads to the scanner.
 2. The firmware file on the scanner is activated (programmed into the scanner). Activation lasts for approximately 50 seconds, during which the LED blinks red. During activation, the scanner does not respond to network queries. When activation (programming) completes, the scanner automatically reboots (the LED turns off) and emits a power up beep, and powers up with the new upgraded 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.

Scenario B: Loading the same version of firmware that is already 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 already on the scanner, while other components are different.
Before firmware loads to the scanner, the scanner driver reads the header information of each firmware component to validate the model number and version. For example, if the first component downloading from the firmware file is the same version as the one already on the scanner, the component does not load to the scanner. Each remaining component in the firmware file is verified against the equivalent component on the scanner, and only components that are different are downloaded to the scanner.

Scenario C: Loading an incompatible version of firmware on the scanner

- This occurs when attempting to load firmware designed for one scanner model say DS6707 onto another incompatible scanner model say 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 scanner driver determines that the firmware component model number does not match the scanner, the component does not load. This process continues to verify each remaining component in the firmware file.

Firmware Upgrade Procedure

1. Connect and discover a scanner (see [Scanner Discovery / Asset Tracking Information / Validating Successful SDK Installation on page 4-6](#)).
2. Obtain the latest firmware .DAT file for loading to a scanner using 123Scan².
 - a. Download and launch 123Scan2 from www.motorola.com/123scan2.
 - b. Using 123Scan², confirm you have the latest scanner plug-in. The plug-in contains a number of files including the firmware file and release notes.
 - i. To download the latest scanner plug-ins from within 123Scan², launch 123Scan², go to the help menu and click **Check for updates**.
 - ii. For a listing of scanner models, plug-ins and firmware files supported in 1123Scan² select **Supported scanners and plug-ins** under the *Help* menu.
 - iii. The plug-ins are contained within a 123Scan² sub folder accessible in: *[WINDOWSDRIVE]\Documents and Settings\[USERNAME]\Application Data\123Scan2\Plug-ins* in Windows XP systems and *[WINDOWSDRIVE]\Users\ Application Data\123Scan2\Plug-ins* in later versions of Windows. The firmware file is named with a .DAT extension (e.g., *CAAABS00-006-R02D0.DAT*).



NOTE If you have the appropriate 123scan² plug-in, extract the firmware .DAT file from plug-in as follows: Rename the file extension of the plug-in file from .SCNPLG to .ZIP and use a standard archive tool, such as WinZip, to extract the firmware update file which ends with the file extension .DAT.

For example the DS9808 plug-in is named *DS9808-COMMON SR MODELS-S-017.SCNPLG*. After changing .SCNPLG to .ZIP, its firmware .DAT file *CAAABS00-006-R02D0.DAT* can be accessed with WinZip.

3. From the *Advanced* tab of the sample application, browse to and select the firmware .DAT file.
4. Check the *Bulk Update* option if bulk channel updating is preferred.



NOTE A firmware update can be performed over one of two possible communication interfaces (channels), USB HID or the much faster USB Bulk. Most SNAPi devices support USB Bulk firmware update but some only support the USB HID channel. To confirm whether or not your scanner supports faster firmware upgrades via USB Bulk mode see [Table 2-2 on page 2-3](#).

5. Click **Update** to transfer the firmware file from the computer to the scanner.
6. Clicking **Update** in the sample application 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 scanner's ID selected in the "Connected Scanners" list of the sample application. The <arg-string> tag contains the path to the firmware file. The <arg-int> tag contains an integer value depending on the selection of bulk update option. If bulk update option is selected, the value of the tag would be "2". Otherwise it would be "1".

7. If you have registered with ScanRMDEvent (see *Register for COM Events on page 5-2*), you receive six types of events per firmware update cycle.
8. The OnScanRMDEvent function has two parameters where the first short type parameter contains the event type described above. The six event type values are listed 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 second parameter of the same function contains an XML for the above event types. By processing the XML further information can be obtained. The formats of the receiving XMLs for each event types are as follows. All XMLs are containing the information about the scanner that 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 scanner driver determines that the firmware component model number does not match the scanners', the component does not load. This process continues to verify each remaining 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 downloads started.

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 it downloading at that moment. The <status> tag contains the status of the download progressing record. 600 value means that it is the resident firmware. For more status and error codes see [Table 3-14 on page 3-32](#).

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>
```

9. After the file transfer is complete, click **Launch** to activate (program into the scanner) the new firmware. Activation takes approximately one minute, during which the LED blinks red and scanning bar code data is disabled. During activation, the scanner does not respond to network queries. When activation (programming) completes, the scanner automatically reboots (the LED turns off), emits a power up beep and restarts with the new upgraded firmware.

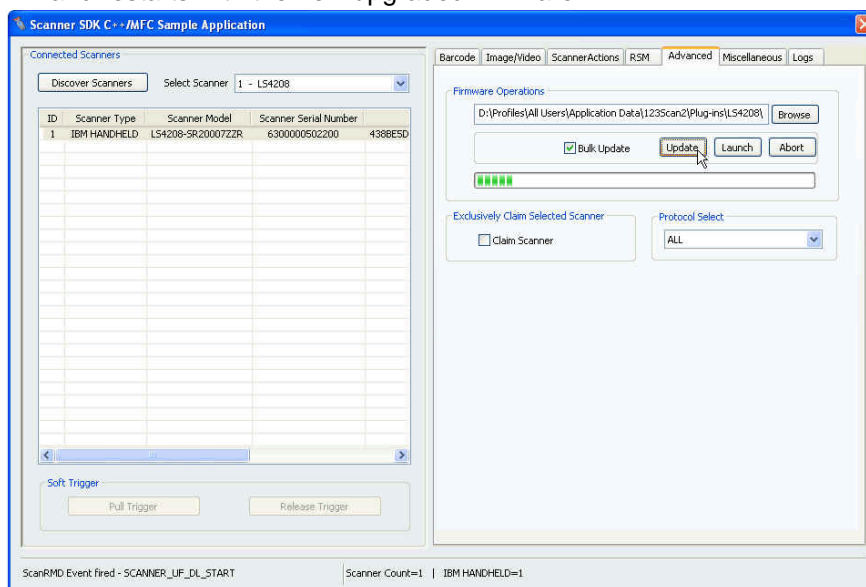


Figure 4-13 Firmware Upgrade Through Bulk (Faster Download Mode) Channel

CHAPTER 5 SAMPLE SOURCE CODE

Overview

This chapter provides information about how a developer uses the Motorola Scanner SDK. This will be demonstrated by code snippets from the sample application.

See [Appendix A, WRITE SIMPLE APPLICATIONS USING CORE SCANNER APIS](#) for a starter example of an application illustrating the Motorola Scanner SDK API. For a list of the most commonly requested topics within this guide, see [Quick Startup](#) in the back of the guide.



NOTE For a complete list of attribute (parameter) numbers and their definitions, download the Attribute Data Dictionary (p/n 72E-149786-xx) from <http://MotorolaSolutions.com/WindowsSDK>. Attributes include configuration parameters, monitored data and asset tracking information.

Sample Utilities Provided in the SDK

The Motorola Scanner SDK includes sample utilities that demonstrate the main functionalities of the SDK. You can gain an understanding of the Motorola Scanner SDK by using these test utilities. In addition, this section describes how to use the utilities functionality.

- Motorola Scanner SDK C++ Sample Application
- Motorola Scanner SDK C# .Net Sample Application

Creation of COM Object And Registration for Events

```
using CoreScanner;  
...  
m_pCoreScanner = new CoreScanner.CCoreScannerClass(); //create COM object  
...
```

Register for COM Events

```
/* Event registration for COM Service */
m_pCoreScanner.ImageEvent += new
CoreScanner._ICoreScannerEvents_ImageEventEventHandler(OnImageEvent);
m_pCoreScanner.VideoEvent += new
CoreScanner._ICoreScannerEvents_VideoEventEventHandler(OnVideoEvent);
m_pCoreScanner.BarcodeEvent += new
CoreScanner._ICoreScannerEvents_BarcodeEventEventHandler(OnBarcodeEvent);
m_pCoreScanner.PNPEvent += new
CoreScanner._ICoreScannerEvents_PNPEventEventHandler(OnPNPEvent);
m_pCoreScanner.ScanRMDEvent += new
CoreScanner._ICoreScannerEvents_ScanRMDEventEventHandler(OnScanRMDEvent);
m_pCoreScanner.CommandResponseEvent += new
CoreScanner._ICoreScannerEvents_CommandResponseEventEventHandler(OnCommandResponseEvent);
m_pCoreScanner.IOEvent += new
CoreScanner._ICoreScannerEvents_IOEventEventHandler(OnIOEvent);
```

Calling Open Command

```
private void Connect()
{
    if (m_bSuccessOpen)
    {
        return;
    }
    int appHandle = 0;
    GetSelectedScannerTypes();
    int status = STATUS_FALSE;

    try
    {
        m_pCoreScanner.Open(appHandle, m_arScannerTypes, m_nNumberOfTypes, out
status);
    }
    ...
}
```

Calling Close Command

```
private void Disconnect()
{
    if (m_bSuccessOpen)
    {
        int appHandle = 0;
        int status = STATUS_FALSE;
        try
        {
            m_pCoreScanner.Close(appHandle, out status);
        }
        ...
    }
}
```

Calling GetScanners Command

```
private void ShowScanners()
{
    lstvScanners.Items.Clear();
    combSlcrScnr.Items.Clear();
    m_arScanners.Initialize();
    if (m_bSuccessOpen)
    {
        m_nTotalScanners = 0;
        short numofScanners = 0;
        int nScannerCount = 0;
        string outXML = "";
        int status = STATUS_FALSE;
        int[] scannerIdList = new int[MAX_NUM_DEVICES];
        try
        {
            m_pCoreScanner.GetScanners(out numofScanners, scannerIdList, out outXML,
out status);
            ...
        }
    }
}
```

Calling ExecCommand Command and ExecCommandAsync Command

```
private void ExecCmd(int opCode, ref string inXml, out string outXml, out int status)
{
    outXml = "";
    status = STATUS_FALSE;
    if (m_bSuccessOpen)
    {
        try
        {
            if (!chkAsync.Checked)
            {
                m_pCoreScanner.ExecCommand(opCode, ref inXml, out outXml, out
status);
            }
            else
            {
                m_pCoreScanner.ExecCommandAsync(opCode, ref inXml, out status);
            }
        }
        ...
    }
}
```


APPENDIX A WRITE SIMPLE APPLICATIONS USING CORE SCANNER APIS

Overview

This appendix provides a step by step guide to writing simple applications using CoreScanner APIs.

Before you start to write applications using CoreScanner APIs, please prepare your development environment properly.

- Install Microsoft Visual Studio 2005 or newer version and make sure you have enough system resources to develop an application on your system.
- Install the Scanner SDK and make sure the SDK is operational. See [How to Verify Scanner SDK Functionality on page 4-6](#) for more information.

Before coding an application in Microsoft C# .Net (console application or an application with a user interface), be prepared with basic reference importing, class declaration and instantiation (see page [A-2](#)).



NOTE For a complete list of attribute (parameter) numbers and their definitions, download the Attribute Data Dictionary (p/n 72E-149786-xx) from <http://MotorolaSolutions.com/WindowsSDK>. Attributes include configuration parameters, monitored data and asset tracking information.

Import CoreScanner Reference, Class Declaration and Instantiation

To create an empty project in Microsoft Visual Studio 2005 (create a console project):

1. Start Microsoft Visual Studio 2005.
2. Go to *File -> New -> Project*

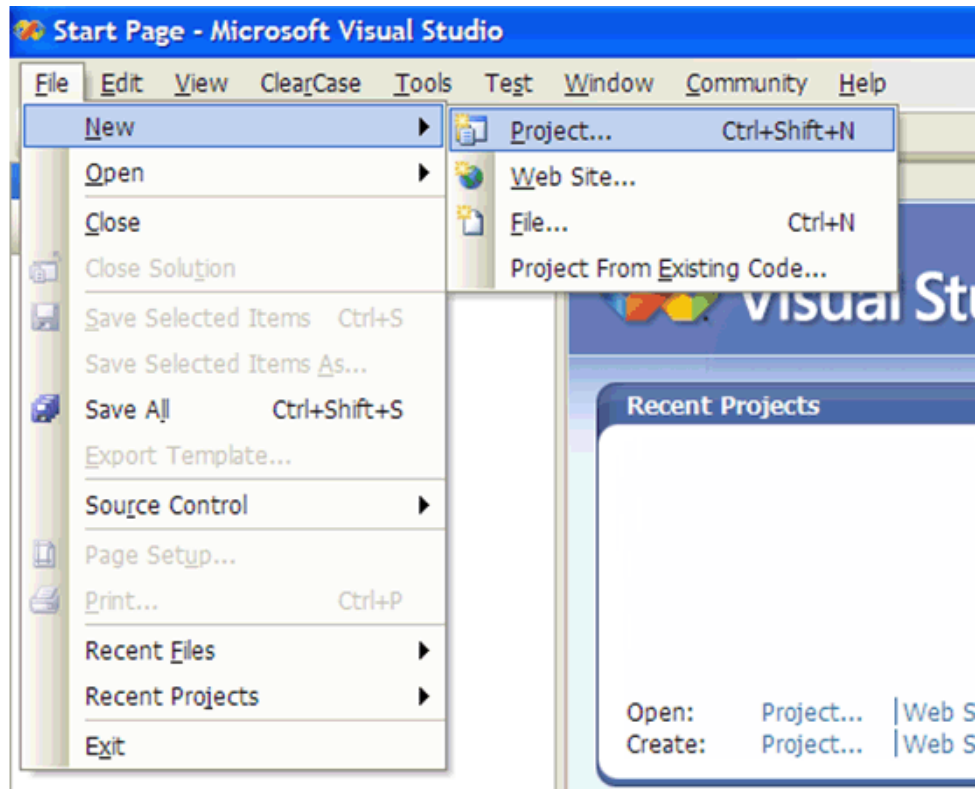


Figure A-1 *New Project*

3. Select Project "Visual C#" and Template as "Console Application" and type a name for your project. In this example, it is "ConsoleApplication1".

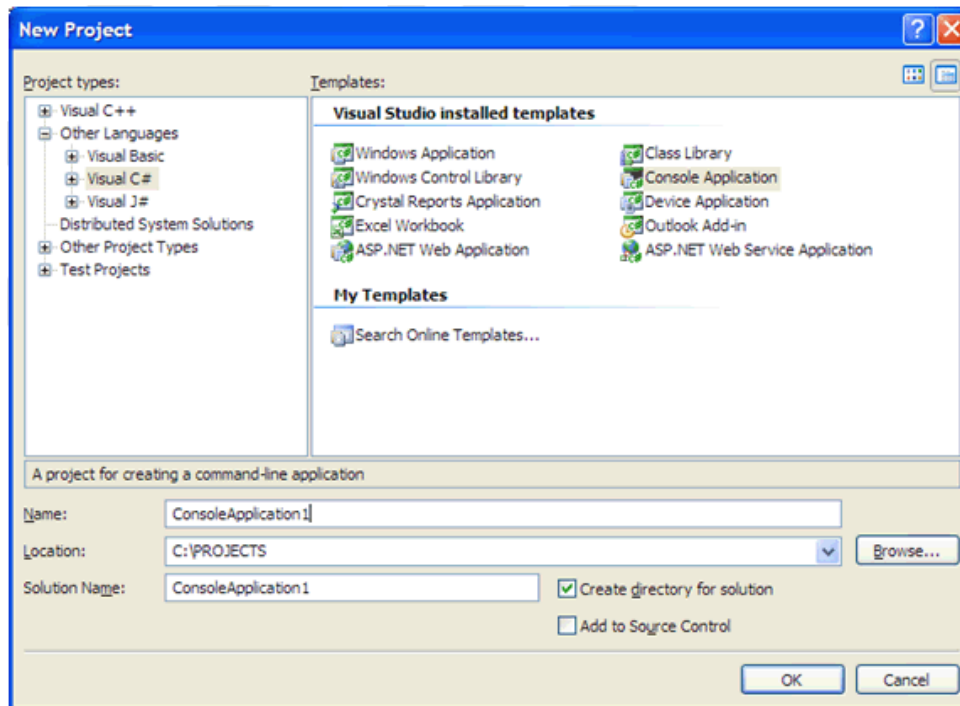


Figure A-2 Console Application1

4. Import CoreScanner as a reference into your application.
5. Go to *Project -> Add Reference...*

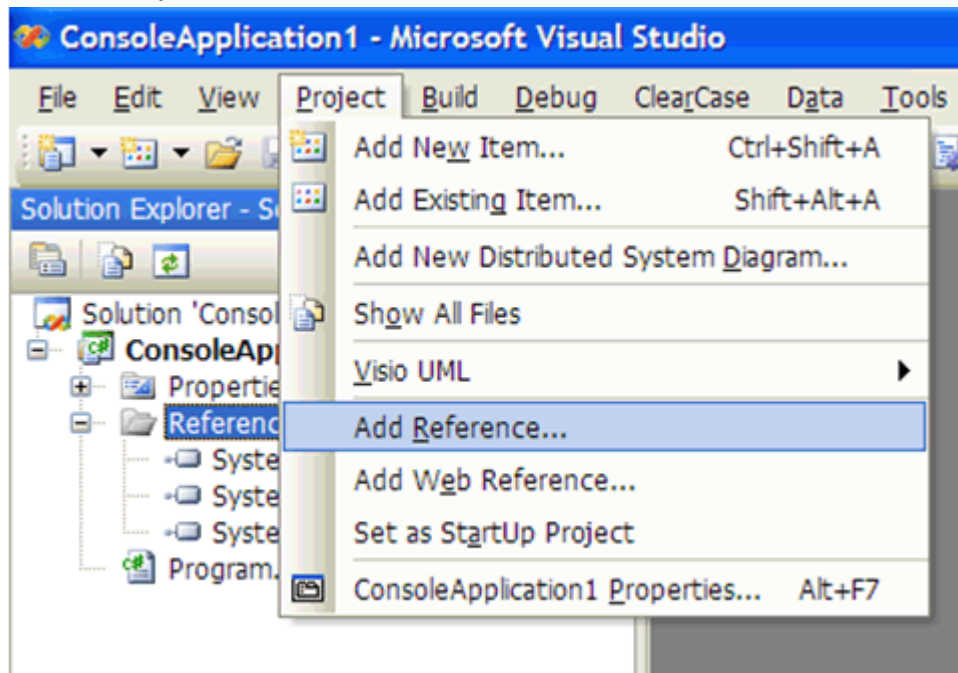


Figure A-3 Add Reference

6. Select the CoreScanner Type Library from the COM tab and click **OK**.

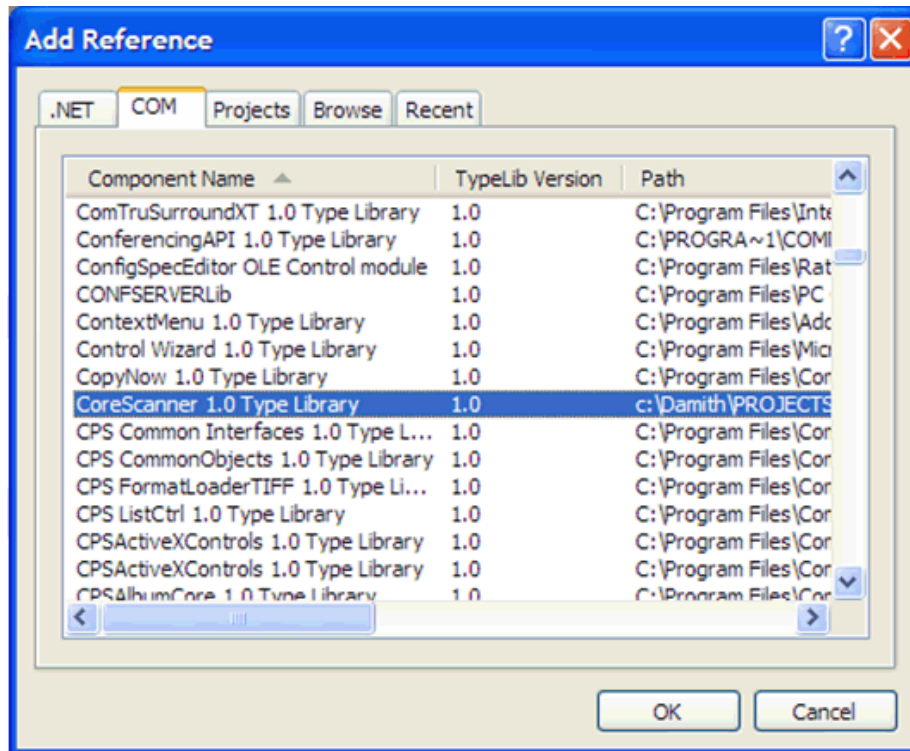


Figure A-4 CoreScanner Type Library

7. CoreScanner is listed in your project under *References* as shown below.

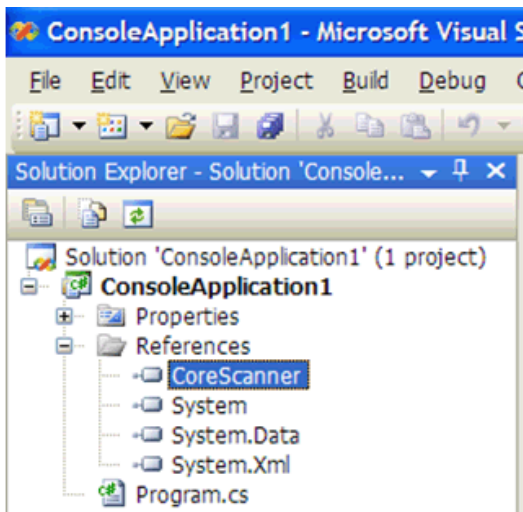


Figure A-5 CoreScanner Reference

8. You are now ready to import the CoreScanner library into your application. After importing, you can declare and instantiate the CoreScanner class for the application.

Open the Program.cs file and enter the modifications as shown below.

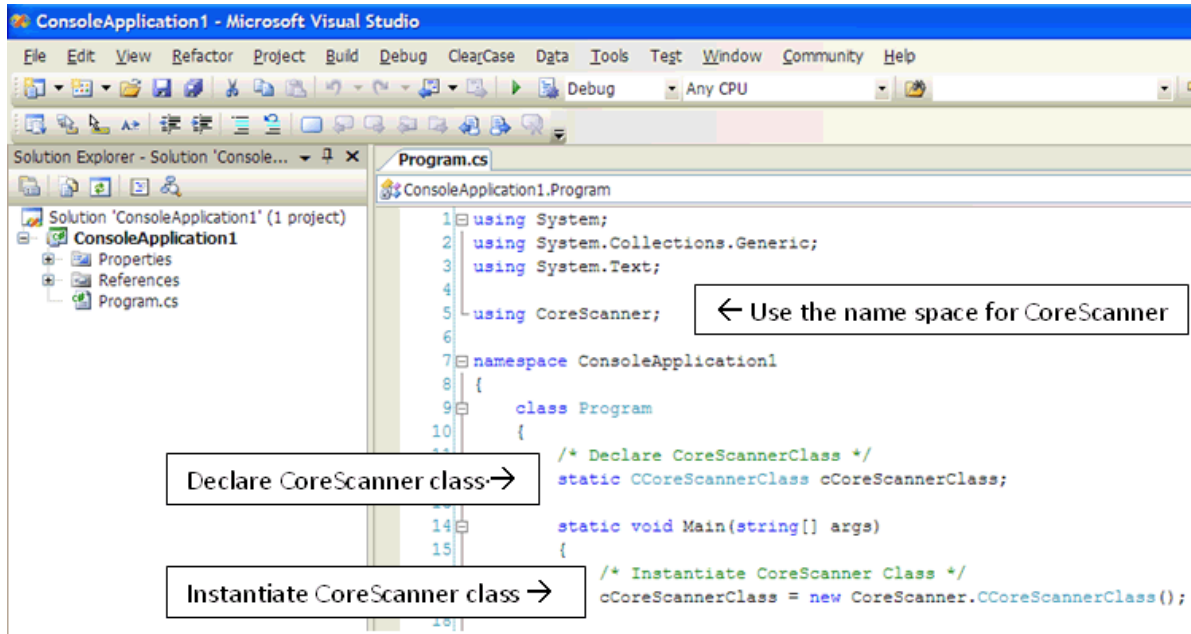


Figure A-6 Open Program.cs File

9. Now you are ready to start dealing with CoreScanner APIs. Follow steps 1 to 7 before you start using APIs.

Call Open API

After you instantiate CoreScanner class into your application you can call Open API as shown below.

```

1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4 using CoreScanner;
5
6 namespace ConsoleApplication1
7 {
8     class Program
9     {
10         /* Declare CoreScannerClass */
11         static CCoreScannerClass cCoreScannerClass;
12
13         static void Main(string[] args)
14         {
15             /* Instantiate CoreScanner Class */
16             cCoreScannerClass = new CoreScanner.CCoreScannerClass();
17
18             /* Call Open API */
19             short[] scannerTypes = new short[1];    // Scanner types you are interested in
20             scannerTypes[0] = 1;                    // 1 for all scanner types
21             short numberOfScannerTypes = 1;          // Size of the scannerTypes array
22             int status;                             // Extended API return code
23
24             cCoreScannerClass.Open(0, scannerTypes, numberOfScannerTypes, out status);
25
26             if (status == 0)
27             {
28                 Console.WriteLine("CoreScanner API: Open successful");
29             }
30             else
31             {
32                 Console.WriteLine("CoreScanner API: Open failed");
33             }
34         }
35     }
36 }

```

Figure A-7 Call Open API Code

If you have successfully executed all the commands, you see the following output on the console window.

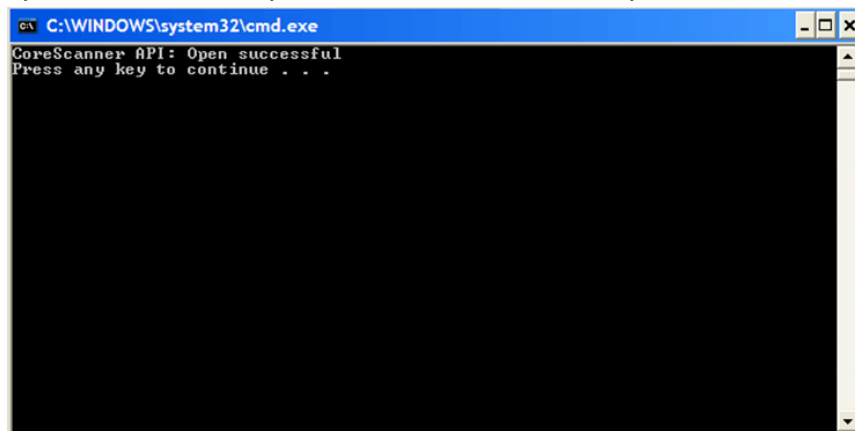


Figure A-8 Open API Success - Console Window

Call GetScanners API

After you call Open API as described on page [A-6](#) you can call the GetScanners API as shown below.

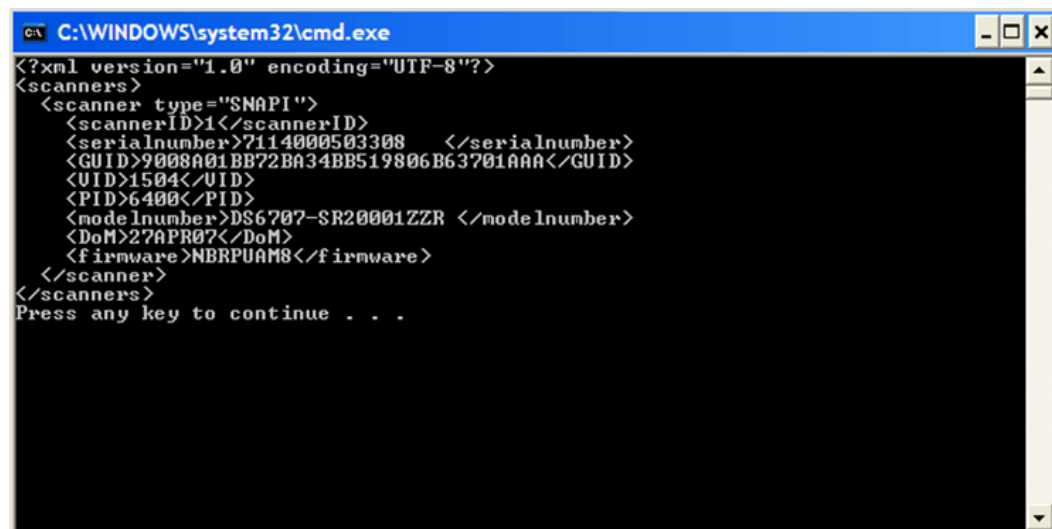
```

1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4 using CoreScanner;
5
6 namespace ConsoleApplication1
7 {
8     class Program
9     {
10         /* Declare CoreScannerClass */
11         static CCoreScannerClass cCoreScannerClass;
12
13         static void Main(string[] args)
14         {
15             /* Instantiate CoreScanner Class */
16             cCoreScannerClass = new CoreScanner.CCoreScannerClass();
17
18             /* Call Open API */
19             short[] scannerTypes = new short[1];    // Scanner types you are interested in
20             scannerTypes[0] = 1;                  // 1 for all scanner types
21             short numberOfScannerTypes = 1;        // Size of the scannerTypes array
22             int status;                            // Extended API return code
23
24             cCoreScannerClass.Open(0, scannerTypes, numberOfScannerTypes, out status);
25
26             /* Lets list down all the scanners connected to the host */
27             short numberOfScanner;                // Number of scanners expect to be used
28             int[] connectedScannerIDList = new int[10]; // List of Scanner IDs to be returned
29             string outXML;                        // Scanner details output
30
31             cCoreScannerClass.GetScanners(out numberOfScanner, connectedScannerIDList, out outXML, out status);
32             Console.WriteLine(outXML);
33
34         }
35     }
36 }
37

```

Figure A-9 Call GetScanners API Code

If you have successfully executed all the commands, you see the following output on the console window.



```

C:\WINDOWS\system32\cmd.exe
<?xml version="1.0" encoding="UTF-8"?>
<scanners>
  <scanner type="SNAPI">
    <scannerID>1</scannerID>
    <serialnumber>7114000503308 </serialnumber>
    <GUID>9008A01BB72BA34BB519806B63701AAA</GUID>
    <UID>1504</UID>
    <PID>6400</PID>
    <modelName>DS6707-SR20001ZZR </modelName>
    <DoM>27APR07</DoM>
    <firmware>NBRPUAM8</firmware>
  </scanner>
</scanners>
Press any key to continue . . .

```

Figure A-10 GetScanners API Success - Console Window

Calling ExecCommand API to Demonstrate Beep the Beeper

After you call Open API as described on page [A-6](#), you can call ExecCommand API as shown below.

```

1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4 using CoreScanner;
5
6 namespace ConsoleApplication1
7 {
8     class Program
9     {
10         /* Declare CoreScannerClass */
11         static CCoreScannerClass cCoreScannerClass;
12
13         static void Main(string[] args)
14         {
15             /* Instantiate CoreScanner Class */
16             cCoreScannerClass = new CoreScanner.CCoreScannerClass();
17
18             /* Call Open API */
19             short[] scannerTypes = new short[1];    // Scanner types you are interested in
20             scannerTypes[0] = 1;                    // 1 for all scanner types
21             short numberOfScannerTypes = 1;          // Size of the scannerTypes array
22             int status;                              // Extended API return code
23
24             cCoreScannerClass.Open(0, scannerTypes, numberOfScannerTypes, out status);
25
26             /* Let beep the beeper */
27             int opcode = 2018;                        // Method for Beep the beeper
28             string outXML;                             // Output
29             string inXml = "<inArgs>" +
30                 "<scannerID>1</scannerID>" +    //The scanner you need to beep
31                 "<cmdArgs>" +
32                 "<arg-int>3</arg-int>" +        //Beep pattern
33                 "</cmdArgs>" +
34                 "</inArgs>";
35
36             cCoreScannerClass.ExecCommand(opcode, ref inXml, out outXML, out status);
37
38         }
39     }
40 }

```

Figure A-11 Call ExecCommand API Code

If you have successfully executed all the commands, you see the following output on the console window. There is no visual output for this beep command but an audible beep sounds from the scanner.

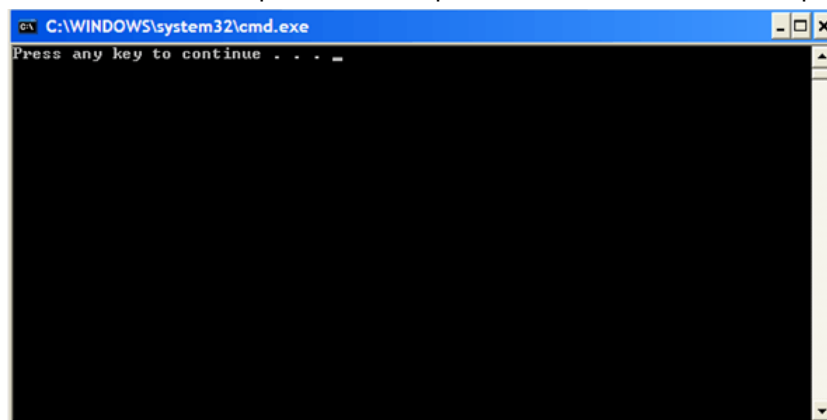


Figure A-12 Call ExecCommand API Success - Console Window

Retrieve Asset Tracking Information from ExecCommand with the RSM_GET Method

After you call Open API as described on page [A-6](#), you can call ExecCommand API as shown below.

```

4 using CoreScanner;
5
6 namespace ConsoleApplication1
7 {
8     class Program
9     {
10         /* Declare CoreScannerClass */
11         static CCoreScannerClass cCoreScannerClass;
12
13         static void Main(string[] args)
14         {
15             /* Instantiate CoreScanner Class */
16             cCoreScannerClass = new CoreScanner.CCoreScannerClass();
17
18             /* Call Open API */
19             short[] scannerTypes = new short[1];    // Scanner types you are interested in
20             scannerTypes[0] = 1;                    // 1 for all scanner types
21             short numberOfScannerTypes = 1;          // Size of the scannerTypes array
22             int status;                              // Extended API return code
23
24             cCoreScannerClass.Open(0, scannerTypes, numberOfScannerTypes, out status);
25
26             /* Lets retrieve assert tracking information */
27             int opcode = 5001;
28             string outXML;                            // XML output
29             string inXml = "<inArgs>" +
30                 "<scannerID>1</scannerID>" +
31                 "<cmdArgs>" +
32                 "<arg-xml>" +
33                 "<attrib_list>20004,533,20007,1</attrib_list>" +
34                 "</arg-xml>" +
35                 "</cmdArgs>" +
36                 "</inArgs>";
37             cCoreScannerClass.ExecCommand(opcode, ref inXml, out outXML, out status);
38             Console.WriteLine(outXML);
39         }
40     }
41 }
42

```

Figure A-13 Retrieve Asset Tracking Information from ExecCommand API Code

If you have successfully executed all the commands, you see the following output on the console window.

```

C:\WINDOWS\system32\cmd.exe
<?xml version="1.0" encoding="UTF-8"?>
<outArgs>
  <scannerID>1</scannerID>
  <arg-xml>
    <modelnumber>DS6707-SR20001ZZR </modelnumber>
    <serialnumber>7114000503308 </serialnumber>
    <GUID>9008A01BB72BA34BB519806B63701AAA</GUID>
    <response>
      <opcode>5001</opcode>
      <attrib_list>
        <attribute>
          <id>20004</id>
          <name></name>
          <datatype>S</datatype>
          <permission>R</permission>
          <value>NBRRPUAM8</value>
        </attribute>
        <attribute>
          <id>533</id>
          <name></name>
          <datatype>S</datatype>
          <permission>R</permission>
          <value>DS6707-SR20001ZZR </value>
        </attribute>
        <attribute>
          <id>20007</id>
          <name></name>
          <datatype>S</datatype>
          <permission>R</permission>
          <value>Imager </value>
        </attribute>
        <attribute>
          <id>1</id>
          <name></name>
          <datatype>F</datatype>
          <permission>RWP</permission>
          <value>True</value>
        </attribute>
      </attrib_list>
    </response>
  </arg-xml>
</outArgs>
Press any key to continue . . . _

```

Figure A-14 Retrieve Asset Tracking Information from ExecCommand Success - Console Window

Enable the UPC-A Attribute by Calling SET_ATTRIBUTE via ExecCommand

After you call Open API as described on page [A-6](#), you can call ExecCommand API as shown below.

```

1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4 using CoreScanner;
5
6 namespace ConsoleApplication1
7 {
8     class Program
9     {
10         /* Declare CoreScannerClass */
11         static CCoreScannerClass cCoreScannerClass;
12
13         static void Main(string[] args)
14         {
15             /* Instantiate CoreScanner Class */
16             cCoreScannerClass = new CoreScanner.CCoreScannerClass();
17
18             /* Call Open API */
19             short[] scannerTypes = new short[1];    // Scanner types you are interested in
20             scannerTypes[0] = 1;                  // 1 for all scanner types
21             short numberOfScannerTypes = 1;         // Size of the scannerTypes array
22             int status;                             // Extended API return code
23
24             cCoreScannerClass.Open(0, scannerTypes, numberOfScannerTypes, out status);
25
26             /* Lets set UPC-A enable/disable */
27             int opcode = 5004;
28             string outXML;                          // XML output
29             string inXml = "<inArgs>" +
30                 "<scannerID>1</scannerID>" +
31                 "<cmdArgs>" +
32                 "<arg-xml>" +
33                 "<attrib_list>" +
34                 "<attribute>" +
35                 "<id>1</id>" +
36                 "<datatype>F</datatype>" +
37                 "<value>True</value>" +
38                 "</attribute>" +
39                 "</attrib_list>" +
40                 "</arg-xml>" +
41                 "</cmdArgs>" +
42                 "</inArgs>";
43             cCoreScannerClass.ExecCommand(opcode, ref inXml, out outXML, out status);
44         }
45     }
46 }
47

```

Figure A-15 Call ExecCommand API Code

This method does not show any output but it sets the UPC-A to enable (True) or disable (False).

Capture Bar Code Data into an Application

1. Create an empty C# *Windows Application* project in Microsoft Visual Studio.

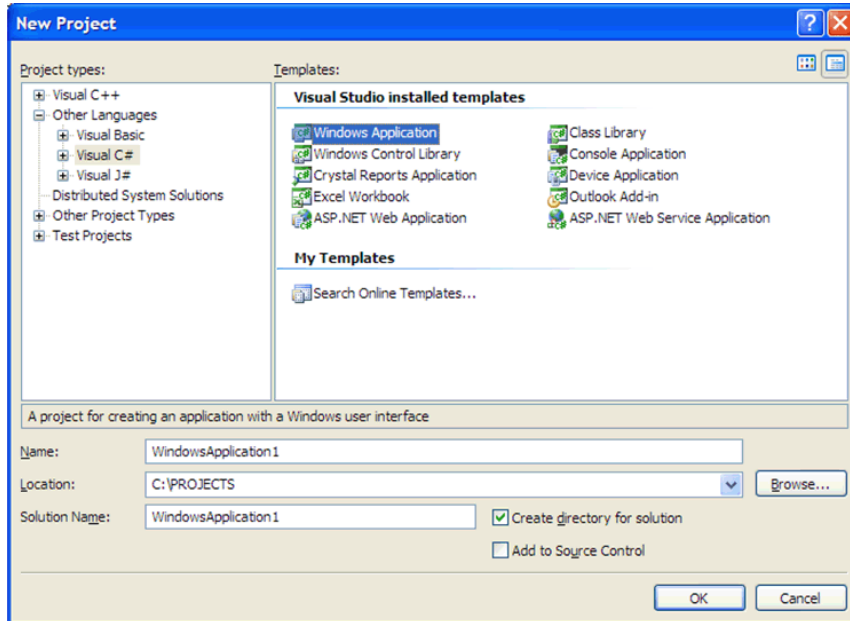


Figure A-16 Create Empty C# Windows Application

2. Add CoreScanner as a reference into your project. See [Import CoreScanner Reference, Class Declaration and Instantiation on page A-2](#) for more details.
3. Add a button and a text area into your application.

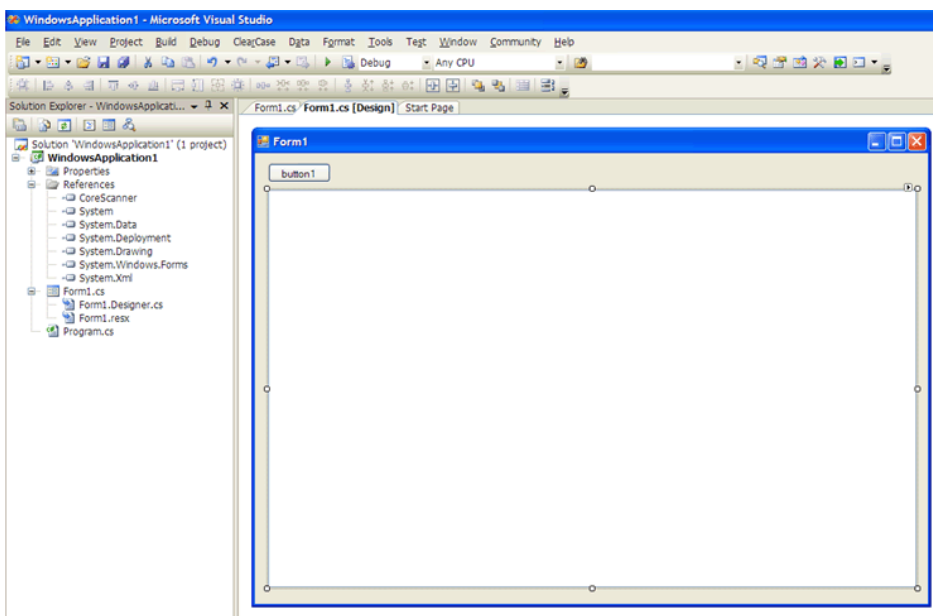


Figure A-17 Add a Button

4. Include the following code segment into the *Button click* method.

```

27 private void button1_Click(object sender, EventArgs e)
28 {
29
30     try
31     {
32         /* Instantiate CoreScanner Class */
33         cCoreScannerClass = new CoreScanner.CCoreScannerClass();
34
35         /* Lets call Open API in CoreScanner */
36         short[] scannerTypes = new short[1];
37         scannerTypes[0] = 1;
38         short numOfScannerTypes = 1;
39         int status;
40
41         cCoreScannerClass.Open(0, scannerTypes, numOfScannerTypes, out status);
42
43         cCoreScannerClass.BarcodeEvent += new CoreScanner._ICoreScannerEvents_BarcodeEventEventHandler(OnBarcodeEvent);
44
45         /* Lets list down all the scanners connected to the host */
46         short numberOfScanners;
47         int[] connectedScannerIDList = new int[10];
48         string outXML;
49
50         cCoreScannerClass.GetScanners(out numberOfScanners, connectedScannerIDList, out outXML, out status);
51
52         /* Lets subscribe to events */
53         int opcode = 1001;
54         string inXml = "<inArgs>" +
55             "<cmdArgs>" +
56             "<arg-int>1</arg-int>" +
57             "<arg-int>1</arg-int>" +
58             "</cmdArgs>" +
59             "</inArgs>";
60
61         cCoreScannerClass.ExecCommand(opcode, ref inXml, out outXML, out status);
62     }
63     catch (Exception exp)
64     {
65         Console.WriteLine("Something wrong please check...." + exp);
66     }
67 }
68

```

← Subscribe to Barcode events on C#

← Subscribe to Barcode events on SDK

Figure A-18 *Button Click Code*

5. Implement a method to receive the event as shown below and populate the text box with scanned data.

```

1 using System;
2 using System.Collections.Generic;
3 using System.ComponentModel;
4 using System.Data;
5 using System.Drawing;
6 using System.Text;
7 using System.Windows.Forms;
8
9 using CoreScanner;
10
11 namespace WindowsApplication1
12 {
13     public partial class Form1 : Form
14     {
15         CCoreScannerClass cCoreScannerClass;
16
17         public Form1()
18         {
19             InitializeComponent();
20         }
21
22         void OnBarcodeEvent(short eventType, ref string scanData)
23         {
24             textBox1.Text = scanData;
25         }
26
27         private void button1_Click(object sender, EventArgs e)
28         {
29
30             try
31             {
32                 /* Instanciate CoreScanner Class */
33                 cCoreScannerClass = new CoreScanner.CCoreScannerClass();
34
35                 /* Lets call Open API in CoreScanner */
36                 _short[] scannerTunes = new short[11];

```

Figure A-19 Method to Receive Event

6. If you execute the application and click on the button, the application instantiates CoreScanner and is ready to receive bar code events. [Figure A-20](#) illustrates the output when you scan a bar code.

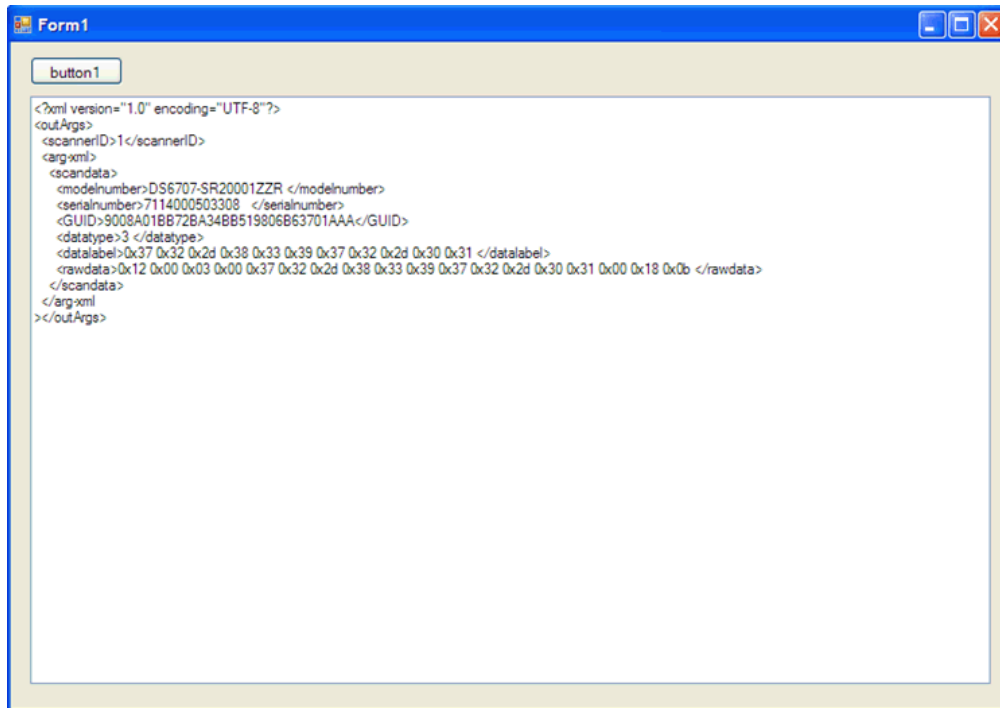


Figure A-20 Scanned Bar Code Output

INDEX

A

APIs

Close	3-4
ExecCommand	3-3
ExecCommandAsync	3-4
GetScanners	3-3
Open	3-2

B

bold text use in guide	x
bullets use in guide	x

C

Close API	3-4
communication modes	
data	1-6
host variant switching	1-5, 1-6, 4-23
image & video	1-6
management	1-6, 1-7
simulated HID keyboard	
output	1-6, 2-11, 4-9, 4-10
components	2-9
conventions	
notational	x

D

data communication mode	1-6
-------------------------	-----

E

error codes	3-32
ExecCommand API	3-3
ExecCommandAsync API	3-4

F

font use in guide	x
-------------------	---

G

GetScanners API	3-3
-----------------	-----

H

host variant switching	
communication mode	1-5, 1-6, 4-23
host variants, supported	2-3

I

image & video communication mode	1-6
information, service	xi
installing SDK	2-4
italics use in guide	x

M

management communication mode	1-6, 1-7
methods	
ABORT_MACROPDF	3-12
ABORT_UPDATE_FIRMWARE	3-12
AIM_OFF	3-12
AIM_ON	3-12
ATTR_GET	3-13
ATTR_GETALL	3-13
ATTR_GETNEXT	3-13
ATTR_SET	3-13
ATTR_STORE	3-13
CLAIM_DEVICE	3-12
DEVICE_BEEP_CONTROL	3-12
DEVICE_CAPTURE_IMAGE	3-12

DEVICE_CAPTURE_VIDEO	3-12
DEVICE_LED_OFF	3-12
DEVICE_LED_ON	3-12
DEVICE_PULL_TRIGGER	3-12
DEVICE_RELEASE_TRIGGER	3-12
DEVICE_SET_PARAMETERS	3-12
DEVICE_SET_SERIAL_PORT_SETTINGS ...	3-13
DEVICE_SWITCH_HOST_MODE	3-13
FLUSH_MACROPDF	3-12
GET_DEVICE_TOPOLOGY	3-13
GET_VERSION	3-12
KEYBOARD_EMULATOR_ENABLE	3-13
KEYBOARD_EMULATOR_GET_CONFIG ...	3-13
KEYBOARD_EMULATOR_SET_LOCALE ...	3-13
REBOOT_SCANNER	3-12
REGISTER_FOR_EVENTS	3-12
RELEASE_DEVICE	3-12
SCAN_DISABLE	3-12
SCAN_ENABLE	3-12
SET_PARAMETER_DEFAULTS	3-12
SET_PARAMETER_PERSISTANCE	3-12
START_NEW_FIRMWARE	3-13
UNREGISTER_FOR_EVENTS	3-12
UPDATE_ATTRIB_META_FILE	3-13

N

notational conventions	x
------------------------------	---

O

Open API	3-2
----------------	-----

R

requirements, system	2-2
----------------------------	-----

S

scanner mode	4-6
SDK components	2-2, 2-9
service informationxi
simulated HID keyboard output	
communication mode	1-6, 2-11, 4-9, 4-10
status codes	3-32
supported host variants	2-3
system requirements	2-2

V

verify SDK functionality	4-6
verifying installation	2-12

Quick Startup

Overview	1-1
Operating systems / System requirements	2-2
Scanner model vs. Communication modes	1-6, 2-3
Block diagram of system	1-5
SDK Components & Installation details	2-1, 2-4
Components and folder paths	2-2, 2-9
Validate SDK installed properly	2-12, 4-6
OPOS / JPOS Drivers	2-1, 2-2, 2-7
WMI / Remote Scanner Management	2-2, 2-4, 2-7, 2-9
Test and sample utilities	4-2
Table of buttons and input fields	4-3
List of utility functionality	4-2
Bar code Data Display	A-12, A-11, 4-8, 4-9
One application connected to two scanners	1-9
Simulated HID Keyboard Output	1-6, 2-11, 4-3, 4-9
Discovery	4-6
Querying asset information	A-9, 4-6, 4-15, 4-17
Query and Set Parameters / Attributes	
Query values	4-15, 4-17, 4-18
Set Value (Device Configuration)	3-12, 4-19
Programming an ADF rule	4-18, 4-21
LED control	4-14, 4-14, 4-22
Beeper control	A-8, 4-13, 4-13, 4-22
Enable / disable a symbology	A-11, 4-20
Capturing an image	4-10
Capturing a video	4-10
Firmware Upgrade	4-24, 4-26
Host Variant Switching	4-24, 4-23
C++ sample application and source code	2-9, 4-2, 5-1
C# sample application and source code	2-9, 4-3, 5-1
Starter application using CoreScanner API	A-1
API overview	3-1
Create com object	5-1
Register for event	5-2, 4-6
Open	A-6, 3-2, 5-2
Get scanner	A-7, 3-3, 4-7, 5-3
Execute command	A-8, 4-15, 3-3, 5-3
List of Methods	3-12
Execute command asynchronously	3-4, 5-3
Close	3-4, 5-2
Attribute Data Dictionary (Index of Parameters)	4-1



MOTOROLA

Motorola Solutions, Inc.
One Motorola Plaza
Holtsville, New York 11742, USA
<http://www.motorolasolutions.com>

MOTOROLA, MOTO, MOTOROLA SOLUTIONS and the Stylized M Logo are trademarks or registered trademarks of Motorola Trademark Holdings, LLC and are used under license. All other trademarks are the property of their respective owners.

© 2011 Motorola Solutions, Inc. All Rights Reserved.



72E-149784-01 Revision A - June 2011

