ZEBRA SCANNER SDK for iOS DEVELOPER GUIDE

ZEBRA SCANNER SDK for iOS DEVELOPER GUIDE

MN001834A04
Revision A
July 2019

Warranty

For the complete Zebra hardware product warranty statement, go to:

http://www.zebra.com/warranty.

Revision History

Changes to the original manual are listed below:

-01 Rev A - 10/2015	Initial Release
-02 Rev A - 12/2015	- Removed all references to sbtEstablishCommunicationSession:aRawPipeMode and all uses of rawPipeMode; - Removed all references to sbtEventRawData - Removed Raw Data Received Event section - Removed the following line in ISbtSdkApiDelegate - - (void) sbtEventRawData:(NSData*)rawData fromScanner:(int)scannerID; - Removed sbtEventRawData section and all references to sbtReadRawData - Removed the following line: (SBT_RESULT) sbtReadRawData:(unsigned char*)buffer maxLength:(int)length bytesRead:(int*)bytesRead forScanner:(int)scannerID; - Removed the sbtReadRawData section - Removed all references to sbtWriteRawData - Remove the following line: (SBT_RESULT) sbtWriteRawData:(unsigned char*)buffer maxLength:(int)length bytesWritten:(int*)bytesWritten forScanner:(int)scannerID; - Removed sbtWriteRawData section. - Updated Beep Control section, table, and code sample - Removed the Turn Off The LED section. - Replaced the entire Turn On the LED section - Added a deprecation statement to sbtBeepControl - Added a deprecation statement to sbtLedControl - Added code to SbtScannerInfo
-03 Rev A - 1/2017	- Updated CS4070 mode to MFI SSI in Table 1-1 - Several updates changing "pulled trigger on scanner ID" to "pulled updated firmware on scanner ID" - Several updates changing "scanner ID 3 to pull the trigger" to "scanner ID 3 to update firmware" - Added BT scanners to Table 1-2 - Added Table 1-3 - Added page motor support and AIM on and AIM off to Performing Operations section - Added sbtSetBTAddress and sbtGetPairingBarcode - Added descriptions for sbtSetBTAddress and sbtGetPairingBarcode - Added tables: Firmware Update Result Code; STC Bar Code Types; STC - Communication Protocol; Set Default Status - Added sbtEventFirmwareUpdate
-04 Rev A - 6/2019	- Added DS8178

Change	Date	Description
-03 Rev A	01/2017	• Updated CS4070 mode to MFI SSI in *Table 1-1* • Several updates changing "pulled trigger on scanner ID" to "pulled updated firmware on scanner ID" • Several updates changing "scanner ID 3 to pull the trigger" to "scanner ID 3 to update firmware" • Added BT scanners to *Table 1-2* • Added *Table 1-3* • Added page motor support and AIM on and AIM off to Performing Operations section • Added sbtSetBTAddress and sbtGetPairingBarcode • Added descriptions for sbtSetBTAddress and sbtGetPairingBarcode • Added tables: Firmware Update Result Code; STC Bar Code Types; STC Communication Protocol; Set Default Status • Added sbtEventFirmwareUpdate
-04 Rev A	07/2019	• Added DS8178 support.

TABLE OF CONTENTS

ABOUT THIS GUIDE

Introduction

The *Zebra Scanner SDK for iOS Developer Guide* provides installation and programming information for the Software Developer Kit (SDK) that allows Software Decode based applications for iOS based devices.

Chapter Descriptions

This guide includes the following topics:

- *Chapter 1, INTRODUCTION to the ZEBRA SCANNER SDK for iOS* provides an overview of the Zebra Scanner SDK for iOS.
- *Chapter 2, ZEBRA SCANNER CONTROL APPLICATION INSTALLATION and CONFIGURATION* describes the process required to install and execute the scanner Zebra Scanner Control application.
- *Chapter 3, SAMPLE SOURCE CODE* provides detailed examples that demonstrate how to develop iOS applications using the Zebra Scanner Software Development Kit (SDK).
- *Chapter 4, ZEBRA SCANNER SDK for iOS API* defines the API that can be used by external applications to connect remote scanners to a specific iOS device and control connected scanners.

Related Documents

- Zebra Scanner SDK for Android Developer Guide, p/n MN002223Axx.
- RFD8500 RFID Developer Guide, p/n MN002222Axx.
- RFD8500 Quick Start Guide, p/n MN002225Axx.
- RFD8500 Regulatory Guide, p/n MN002062Axx.
- CRDUNIV-RFD8500-1R Three Slot Universal Cradle Charge Only Regulatory Guide, p/n MN002224Axx.
- RFD8500 User Guide, p/n MN002065Axx.

For the latest version of this guide and all guides, go to: www.zebra.com/support.

Notational Conventions

This document uses the following conventions:

- The prefix SBT is used to reference Zebra Scanner SDK for iOS APIs.
- The abbreviation for Bluetooth is BT.
- SDK refers to the Zebra Scanner SDK for iOS.
- *Italics* are used to highlight chapters, screen names, fields, and sections in this and related documents
- 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.

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

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

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

Service Information

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

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

- Product name
- Version number

Zebra 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 Zebra support, you may need to return your equipment for servicing and will be given specific directions. Zebra 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 Zebra business partner, contact that business partner for support.

Chapter 1 INTRODUCTION to the ZEBRA SCANNER SDK for iOS

Overview

The Zebra Scanner SDK for iOS is provided as a static library that can be linked to a custom iOS application. The static library provides an Application Programming Interface (API) to provide the communication facilities required to operate a supported Zebra scanner on the iOS platform. The SDK provides the capability to manage connections, configure connected scanners, perform various operations with connected scanners, and to obtain information about connected scanners.

Supported Scanners

The following MFi (Made for iPhone/iPod/iPad) Zebra scanners are currently supported by the iOS SDK.

Table 1-1 *Supported Zebra Scanners*

Zebra Scanner	Configuration
CS4070	MFI SSI
RFD8500	Bluetooth MFi Server mode, with primary mode set to MFI SSI

Table 1-2 lists the Bluetooth low energy Zebra scanners currently supported by the iOS SDK.

Table 1-2 *Supported Zebra Scanners*

Zebra Scanner	Configuration
DS3678	SSI over Bluetooth low energy
LI3678	SSI over Bluetooth low energy
DS8178	SSI over Bluetooth low energy

✓ ***NOTE*** To configure a device in the mode specified, refer to the appropriate Product Reference Guide, User Guide, or Integration Guide.

Table 1-3 *Pairing Modes*

Scanner Model	Pairing Bar Code		
	Legacy	STC	Manual Pairing
LI3678	No	Yes	Yes
DS3678	No	Yes	Yes
CS4070	No	No	Yes
RFD8500	No	No	Yes
DS8178	Yes	Yes	Yes

System Requirements

The following system requirements are necessary in order to develop, test, and execute applications using the Zebra Scanner SDK for iOS:

- Xcode 6, or later
- iOS SDK 8.0, or later
- Zebra Scanner SDK for iOS
- iPhone or iPod Touch running iOS 8.0 or later.

Chapter 2 ZEBRA SCANNER CONTROL APPLICATION INSTALLATION and CONFIGURATION

Introduction

This chapter describes the process required to install and execute the Zebra Scanner Control application that is provided with the Zebra Scanner SDK for iOS.

✓ ***NOTE*** The purpose of the scanner Zebra Scanner Control application is to demonstrate the various capabilities of the SDK library. It is not intended for production use purposes.

Installing the Zebra Scanner Control Application

Using the Source Code

Prerequisites

- OS X.
- Xcode. Download Xcode from the Mac App Store.
- iOS 8, or higher, installed.
- An iOS Developer account with Apple.
- A provisioned iOS device.

Launching the Zebra Scanner Control Application

To launch the application on the device:

1. Connect a provisioned iOS device to your Mac.
2. Open the *ScannerSDKApp.xcodeproj* project file.
3. In the project navigator, choose your device from the *Scheme* toolbar menu.

✓ ***NOTE*** If your iOS device is listed as an ineligible device, fix the issue before continuing. The version of iOS that is installed on the device must match the deployment target set in the project.

4. Click the **Run** button. Xcode installs the application on the device.
5. If a warning message appears that states *No matching provisioning profile found*, click the **Fix Issue** button. Xcode needs to add the device to the team provisioning profile before it can launch the application on the device.
6. If a prompt appears that asks whether or not *codesign* may sign the application using a key in your *keychain*, click **Always Allow**.
7. The application launches on your device.

From the Apple App Store

Prerequisites

- An iOS device with iOS 8, or higher, installed.

Launching the Zebra Scanner Control Application

To launch the scanner application on the device from the Apple App Store:

1. Launch the App Store application on your iOS device.
2. Search for the *Zebra Scanner* application, or *Zebra Scanner Control* application by Zebra Technologies.
3. Tap the **Get** button next to the application, then tap the **Install** button.

Launching the Zebra Scanner Control Application

1. Before launching the application, set the Zebra scanner to the appropriate configuration. See *Supported Scanners on page 1-1* to determine the required configuration needed for your Zebra scanner.
2. Enable Bluetooth on your iOS device and pair it with the Bluetooth Zebra scanner, if it is not paired already. Refer to the relevant Product Reference Guide, User Guide, or Integration Guide for instructions on how to pair the device.
3. Launch the application by selecting the Zebra Scanner Control application from your device's home screen. The following screen displays:

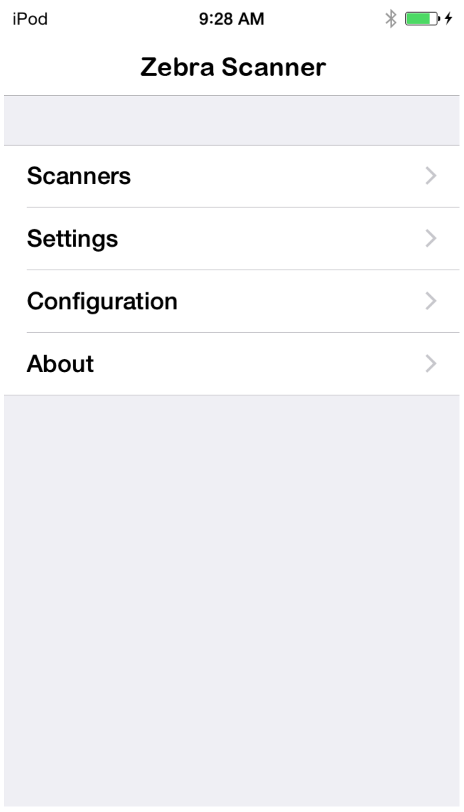
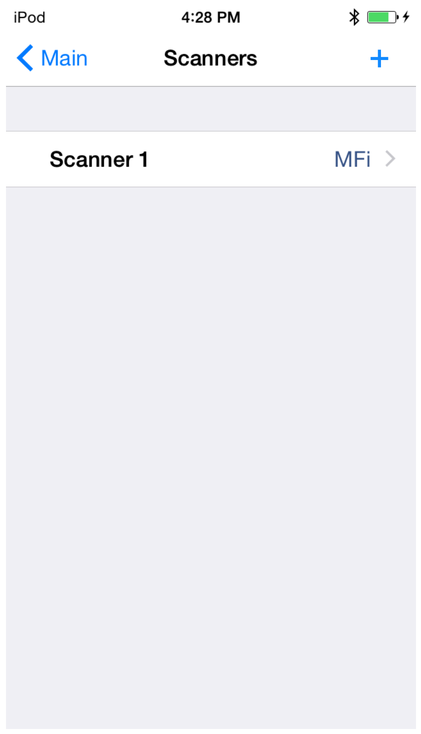


Figure 2-1 *Scanner Application - Initial Screen*

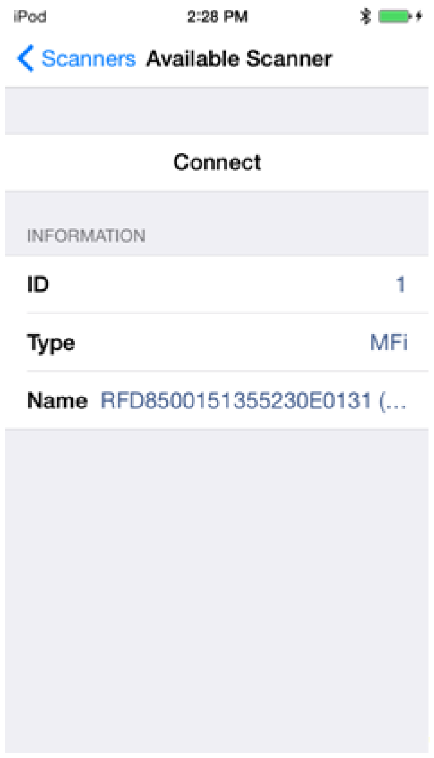
4. Select *Scanners* from the menu. The following screen displays:



Scanner 1 MFi

Figure 2-2 *Figure 2 - Scanner Application - Scanners Screen*

5. Select the device from the left panel, and then tap the **Connect** button from the *Available Scanner* menu.The following screen displays:



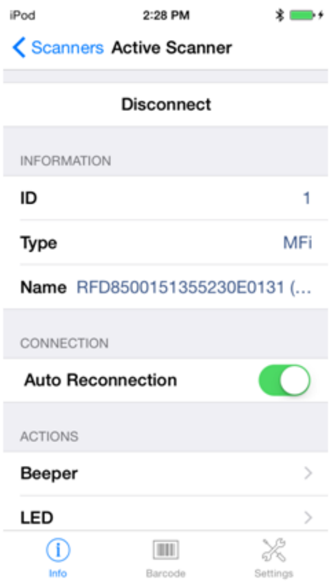
INFORMATION

ID	1
Type	MFi
Name	RFD8500151355230E0131 (...

Figure 2-3 *Scanner Application - Available Scanner Screen*

6. Once connected, the *Active Scanner* menu displays. This menu is used to select various options including *Info*, *Barcode*, and *Settings*.

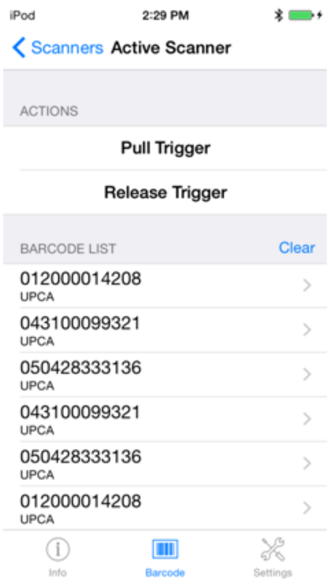
a. Select the *Info* tab to view the *Scanner ID*, *Type*, *Name*, and *Auto Reconnection* state. This tab also contains functionality to control the Beeper and LED. The following screen displays:



ID	1
Type	MFi
Name	RFD8500151355230E0131 (...

Figure 2-4 *Scanner Application - Active Scanner Screen*

b. Select the *Barcode* tab to view scanned bar codes, and to pull/release the scanner trigger.The following screen displays:



012000014208
UPCA
043100099321
UPCA
050428333136
UPCA
043100099321
UPCA
050428333136
UPCA
012000014208
UPCA

Figure 2-5 *Scanner Application - Barcode screen*

c. Select the *Settings* tab to select supported Symbologies, configure the beeper volume and frequency, and to enable/disable scanning.The following screen displays:

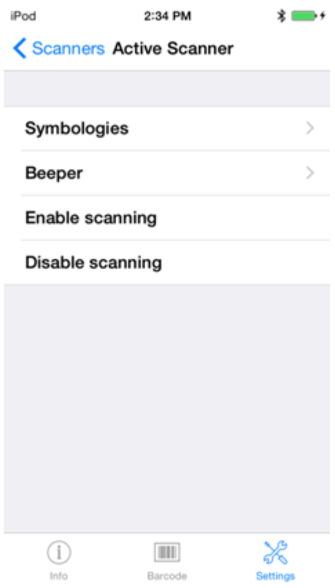
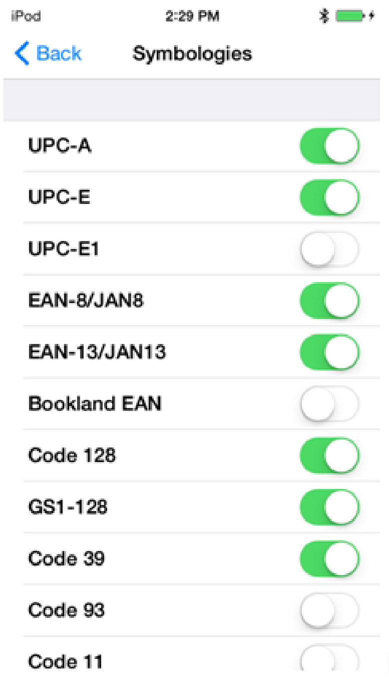


Figure 2-6 *Scanner Application - Settings Screen*

i. To control which symbologies the scanner reads, select the *Symbologies* option. An on/off control switch appears next to each bar code type displayed in the menu.



UPC-A ●
UPC-E ●
UPC-E1 ○
EAN-8/JAN8 ●
EAN-13/JAN13 ●
Bookland EAN ○
Code 128 ●
GS1-128 ●
Code 39 ●
Code 93 ○
Code 11 ○

Figure 2-7 *Scanner Application - Symbologies Screen*

✓ ***NOTE*** Custom symbology settings can be configured via 123Scan^{2}. When setting the Inter Character Delay for HID only, set the delay in milliseconds between emulated keystrokes. When pairing with an Android device, set the Inter Character Delay to 70 ms to avoid data loss.

Configuring the Xcode Project

The Zebra Scanner SDK for iOS consists of a static library that can be linked with an external iOS application and a set of necessary header files.

To add the SDK:

1. Create a new Xcode project and save it to a new project folder.
2. Copy the symbolbt-sdk folder provided by Zebra Technologies into the new project folder. This folder contains the libsymbolbt-sdk.a file and an include folder containing SDK header files.
3. Open the new project in Xcode, and select your project in the file navigator sidebar.
4. Configure the Xcode project to support one or more external accessory communication protocols through the UISupportedExternalAccessoryProtocols key in your application *Info.plist* file or via the Info tab of your project settings.

Table 2-1 *Communication Protocols*

Communication Protocol	Zebra Scanner
com.motorolasolutions.CS4070_ssi	CS4070
com.zebra.scanner.SSI com.motorolasolutions.scanner	RFD8500

5. In order to configure your application to communicate with Bluetooth scanners in a background mode, configure your Xcode project to specify the background modes that your application supports using the UIBackgroundModes key in your application *Info.plist* file or via the Info tab of your project settings.
6. Select your *Target*, and then its *Build Phases* tab.
7. Expand the *Link Binary With Libraries* item.
8. Add the following frameworks by clicking the **+** button:
   - libsymbolbt-sdk.a
   - ExternalAccessory
   - CoreBluetooth.
9. Select the *Build Settings* tab.
10. Navigate to the *Search Paths* section of the *Build Settings*.
11. Set *User Header Search Paths* to *$(SRCROOT)/symbolbt-sdk/include/*.
12. Your project is now configured to use the Zebra Scanner SDK for iOS.

Chapter 3 SAMPLE SOURCE CODE

Introduction

This chapter provides detailed examples that demonstrate how to develop iOS applications using the Zebra Scanner Software Development Kit (SDK).

Accessing the SDK

Create an instance of the Zebra Scanner SDK for iOS API inside of a class that conforms to the `ISbtSdkApiDelegate`. All available API functions are defined by the `ISbtSdkApi` Objective-C protocol. A single shared instance of an API object that implements the ISbtSdkApi can be obtained via `createSbtSdkApiInstance` method of the `SbtSdkFactory` class.

Code Snippet - Obtain Instance of API

```
// ....
@import "SbtSdkFactory.h"

// ....

// Get instance to the Zebra Scanner SDK API
id <ISbtSdkApi> apiInstance = [SbtSdkFactory createSbtSdkApiInstance];

// Get the SDK version string
NSString *version = [apiInstance sbtGetVersion];

NSLog(@"Zebra SDK version: %@\n", version);
```

Event Handling

The SDK supports a set of asynchronous notifications to inform the application about scanner related events. This includes connectivity related events (i.e., appearance of a scanner), and scanner action events (i.e., received bar code data). All supported callbacks are defined by the `ISbtSdkApiDelegate` Objective-C protocol.

In order to receive asynchronous notifications from the SDK, the application performs the following steps:

1. Create an object that implements the `ISbtSdkApiDelegate`.

   Code Snippet - Create A Class That Implements Isbtsdkapidelegate

   ```
   // Definition of a class that implements the ISbtSdkApiDelegate protocol
   @interface EventReceiver : NSObject <ISbtSdkApiDelegate> {

   // TODO: variables

   }

   // TODO: methods definition
   ```

2. Register the created object as notification receiver via the `sbtSetDelegate` function.

   Code Snippet - Set Delegate Of API Instance To Object That Implements ISbtSdkApiDelegate

   ```
   // Registration of the callback interface with SDK
   EventReceiver *eventListener = [[EventReceiver alloc] init];

   [apiInstance sbtSetDelegate:eventListener];
   ```

3. Subscribe for specific asynchronous events using the `sbtSubscribeForEvents` method to specify which events should be reported by the SDK.

   Valid notification/event mask values include:
   - SBT_EVENT_BARCODE
   - SBT_EVENT_IMAGE
   - SBT_EVENT_VIDEO
   - SBT_EVENT_SCANNER_APPEARANCE
   - SBT_EVENT_SCANNER_DISAPPEARANCE
   - SBT_EVENT_SESSION_ESTABLISHMENT
   - SBT_EVENT_SESSION_TERMINATION
   - SBT_EVENT_RAW_DATA.

Code Snippet - Subscribe To Scanner Events

```
// Subscribe to scanner appearance/disappearance, session establishment/termination,
// barcode, and image & video event notifications.
[apiInstance sbtSubsribeForEvents:SBT_EVENT_SCANNER_APPEARANCE |
SBT_EVENT_SCANNER_DISAPPEARANCE | SBT_EVENT_SESSION_ESTABLISHMENT |
SBT_EVENT_SESSION_TERMINATION | SBT_EVENT_BARCODE | SBT_EVENT_IMAGE |
SBT_EVENT_VIDEO];
```

Events

If an object is registered as a notification receiver, the SDK calls the corresponding method of the registered object when a particular event occurs and the application is subscribed for events of this type.

Scanner Appeared Event

This event occurs when the presence of a scanner appears.

Code Snippet - Scanner Appeared Event

```
- (void)sbtEventScannerAppeared:(SbtScannerInfo*)availableScanner
{
    // TODO: Handle event

}
```

Scanner Disappeared Event

This event occurs when a scanner is no longer present.

Code Snippet - Scanner Disappeared Event

```
- (void) sbtEventScannerDisappeared:(int)scannerID
{
    // TODO: Handle event

}
```

Communication Session Established Event

This event occurs when communication is established with a scanner.

Code Snippet - Scanner Session Established Event

```
   - (void)
sbtEventCommunicationSessionEstablished:(SbtScannerInfo*)activeScanner
   {
       // TODO: Handle event

   }
```

Communication Session Terminated Event

This event occurs when communication with a scanner is terminated.

Code Snippet - Scanner Session Terminated Event

```
   - (void) sbtEventCommunicationSessionTerminated:(int)scannerID
   {
       // TODO: Handle event

   }
```

Bar Code Information Received Event

This event occurs when bar code data is read and received.

Code Snippet - Bar Code Data Received Event

```
   - (void) sbtEventBarcodeData:(NSData *)barcodeData
barcodeType:(int)barcodeType fromScanner:(int)scannerID
   {
       // TODO: Handle event

   }
```

Image Data Received Event

This event occurs when image data is received.

Code Snippet - Image Data Received Event

```
   - (void) sbtEventImage:(NSData*)imageData fromScanner:(int)scannerID
   {
       // TODO: Handle event

   }
```

Video Data Received Event

This event occurs when video data is received.

Code Snippet - Video Data Received Event

```
   - (void) sbtEventVideo:(NSData*)videoFrame fromScanner:(int)scannerID
   {
       // TODO: Handle event

   }
```

Firmware Update Event

This event occurs when firmware update is in progress. You don't need to specifically subscribe to this event. You just have to implement this delegate method.

Code Snippet - Firmware Update Event

```
    -  (void) sbtEventFirmwareUpdate:(FirmwareUpdateEvent*)event
    {
    // TODO: Handle event
}
```

Connectivity Management

The SDK identifies scanners as "available" and "active". An "available" scanner is a scanner that is connected to the iOS device via Bluetooth, but has not yet established a communication session with the SDK. An "active" scanner is a scanner that is both connected to the iOS device via Bluetooth, and has established a communication session with the SDK.

The SDK supports simultaneous interaction with multiple scanners. To distinguish between various scanners, the SDK assigns a unique integer identifier for each scanner when it becomes available for the first time.

Set Operation Mode

Zebra Scanner SDK for iOS is designed to support interaction with scanners operating in BT MFi or BT LE mode. The SDK shall be configured to enable communication with a particular type of scanner by setting the operation mode.

Use the `sbtSetOperationalMode` function to set the required operational mode of the scanners to interface with.

Valid inputs include:

- `SBT_OPMODE_MFI`
- `SBT_OPMODE_BTLE`
- `SBT_OPMODE_ALL`.

Code Snippet - Set Operational Mode

```
    // ....

    // Set operational mode to all so that SDK can interface with scanners
    // operating in MFI or BTLE mode.
    [apiInstance sbtSetOperationalMode:SBT_OPMODE_ALL];
```

Enable Scanner Detection

The SDK supports automatic detection of appearance and disappearance of available scanners. When the "Available scanners detection" option is enabled the SDK updates its internal list of available scanners and deliver a corresponding asynchronous notification once it detects connection or disconnection of a particular scanner to the iOS device via Bluetooth. If the option is disabled, the SDK updates its internal list of available scanners only when requested by the application via the `sbtGetAvailableScannersList`.

Use the `sbtEnableAvailableScannersDetection` method to actively detect appearance and disappearance of scanners.

Code Snippet - Enable Auto Detection

```
   // Actively detect appearance/disappearance of scanners
   [apiInstance sbtEnableAvailableScannersDetection:YES];
```

Get Available and Active Scanners

The SDK maintains an internal list of active and available scanners. The following example demonstrates how to receive a list of available and active scanners from the SDK.

Code Snippet - Get Available And Active Scanners

```
   // ....

   // Allocate an array for storage of a list of available scanners
   NSMutableArray *availableScanners = [[NSMutableArray alloc] init];

   // Allocate an array for storage of a list of active scanners
   NSMutableArray *activeScanners = [[NSMutableArray alloc] init];

   // Retrieve a list of available scanners
   [apiInstance sbtGetAvailableScanners:&availableScanners];

   // Retrieve a list of active scanners
   [apiInstance sbtGetActiveScanners:&activeScanners];

   // Merge the available and active scanners into a single list
   NSMutableArray *allScanners = [[NSMutableArray alloc] init];
   [allScanners addObjectsFromArray:availableScanners];
   [allScanners addObjectsFromArray:activeScanners];

   // Print information for each available and active scanner
   for (SbtScannerInfo *info in allScanners)
   {
       NSLog(@"Scanner is %@: ID = %d name = %@", (([info isActive] == YES) ?
@"active" : @"available"), [info getScannerId], [info getScannerName]);

   }
```

Connect to a Scanner

You can establish a connection with an available scanner by providing the scanner ID to the SDK's `sbtEstablishCommunicationSession` method.

Code Snippet - Connect To A Scanner

```
// ....

// Attempt to establish a connection with the scanner device that has an ID = 3.
SBT_RESULT result = [apiInstance sbtEstablishCommunicationSession:3];

if (result == SBT_RESULT_SUCCESS)
{
    // TODO: Process successful connection
    NSLog(@"Connection to scanner ID %d successful",scannerId);
}
else
{
    // TODO: Process error (unsuccessful connection)
    NSLog(@"Failed to establish a connection with scanner ID %d",scannerId);
}
```

Disconnect from a Scanner

You can disconnect from an active scanner using the SDK's `sbtTerminateCommunicationSession` method.

Code Snippet - Disconnect From A Scanner

```
// ....

// Attempt to disconnect from the scanner device that has an ID = 3.
SBT_RESULT result = [apiInstance sbtTerminateCommunicationSession:3];

if (result == SBT_RESULT_SUCCESS)
{
    // TODO: Process successful disconnection
    NSLog(@"Disconnect from scanner ID %d successful",scannerId);
}
else
{
    // TODO: Process error (unsuccessful connection)
    NSLog(@"Failed to disconnect from scanner ID %d",scannerId);
}
```

Automatically Reconnect to Scanner

The SDK supports "Automatic Session Reestablishment" option is used to automatically reconnect to a scanner that unexpectedly disappeared. If the "Automatic Session Reestablishment" option is enabled for a specific scanner, the SDK automatically attempts to reestablish a connection with the scanner when it becomes available again. The option has no effect if the application has intentionally terminated a communication session with the scanner via the `sbtTerminateCommunicationSession` function.

Use the `sbtEnableAutomaticSessionReestablishment` function to enable or disable automatic reconnection for a specific scanner.

Code Snippet - Enable Automatic Reconnection

```
   // ....

   // Set the automatic session reestablishment option for scanner ID 3
   SBT_RESULT result = [apiInstance sbtEnableAutomaticSessionReestablishment:YES 
forScanner:3];

   if (result == SBT_RESULT_SUCCESS)
   {
       // TODO: Option successfully set
       NSLog(@"Automatic Session Reestablishment for scanner ID %d has been set 
successfully",scannerId);
   }
   else
   {
       // TODO: Error (option was not set)
       NSLog(@"Automatic Session Reestablishment for scanner ID %d could not be 
set",scannerId);
   }
```

Performing Operations

The Zebra Scanner iOS SDK provides the capability to perform various scanner operations such as:

- Adjusting beeper settings
- Turning the LED on and off
- Enabling and disabling scanning
- Enabling and disabling supported symbologies
- Pulling and releasing the trigger
- Turning AIM on and off
- Activate page motor.

Beeper

Get Beeper Settings

The get attribute command (SBT_RSM_ATTR_GET) is used to retrieve the beeper volume and frequency attribute values. The attribute values are returned from the scanner as an XML formatted string.

Table 3-1 *Beeper Setting Attributes*

Attribute Description	Attribute Name
Beeper Volume	RMD_ATTR_BEEPER_VOLUME
Beeper Frequency	RMD_ATTR_BEEPER_FREQUENCY

The following code example demonstrates how to retrieve the scanner's current beeper volume and frequency settings.

Code Snippet - Retrieve Beeper Settings

```
   // ....

   // Create XML string to request beeper volume and frequency settings for
scanner ID 3
   NSString *xmlInput = [NSString stringWithFormat:@"
<inArgs><scannerID>%d</scannerID><cmdArgs><arg-xml><attrib_list>%d,%d</attrib_lis
t></arg-xml></cmdArgs></inArgs>",3, RMD_ATTR_BEEPER_VOLUME,
RMD_ATTR_BEEPER_FREQUENCY];

   NSMutableString *xmlResponse = [[NSMutableString] alloc] init];

   // Retrieve beeper volume and frequency settings from Scanner ID 3
   SBT_RESULT result = [apiInstance sbtExecuteCommand:SBT_RSM_ATTR_GET
aInXML:xmlInput aOutXML:&xmlResponse forScanner:3];

   if (result == SBT_RESULT_SUCCESS)
   {
       // TODO: Process xml string containing requested beeper settings
       NSLog(@"Retrieved beeper settings from scanner ID %d:
%@",scannerId,xmlResponse);
   }
   else
   {
       // TODO: Process error
       NSLog(@"Failed to retrieve beeper settings from scanner ID %d",scannerId);
   }
```

Set Beeper Volume

The set attribute command (SBT_RSM_ATTR_SET) is used to set the value of the beeper volume attribute (RMD_ATTR_BEEPER_VOLUME). *Table 3-2* includes valid volume attribute values.

Table 3-2 *Beeper Volume Attribute Values*

Beeper Volume Attribute Values
RMD_ATTR_VALUE_BEEPER_VOLUME_LOW
RMD_ATTR_VALUE_BEEPER_VOLUME_MEDIUM
RMD_ATTR_VALUE_BEEPER_VOLUME_HIGH

The following code example demonstrates how to set the scanner's beeper volume;

Code Snippet - Set The Beeper Volume

```
   // ....

   // Create XML string to set the beeper volume on scanner ID 3 to LOW.
   NSString *xmlInput = [NSString stringWithFormat:@"
<inArgs><scannerID>%d</scannerID><cmdArgs><arg-xml><attrib_list><attribute><id>%d
</id><datatype>B</datatype><value>%d</value></attribute></attrib_list></arg-xml><
/cmdArgs></inArgs>",3, RMD_ATTR_BEEPER_VOLUME, RMD_ATTR_VALUE_BEEPER_VOLUME_LOW];

   // Command Scanner ID 3 to set beeper volume to LOW.
   SBT_RESULT result = [apiInstance sbtExecuteCommand:SBT_RSM_ATTR_SET
aInXML:xmlInput aOutXML:nil forScanner:3];

   if (result == SBT_RESULT_SUCCESS)
   {
       // TODO: Process successful operation
       NSLog(@"Successfully updated beeper settings for scanner ID %d:
%@",scannerId,xmlResponse);
   }
   else
   {
       // TODO: Process error
       NSLog(@"Failed to update beeper settings from scanner ID %d",scannerId);
   }
```

Set Beeper Frequency

The set attribute command (SBT_RSM_ATTR_SET) is used to set the value of the beeper frequency attribute (RMD_ATTR_BEEPER_FREQUENCY). *Table 3-3* includes valid frequency attribute values.

Table 3-3 *Beeper Frequency Attribute Values*

Beeper Frequency Attribute Values
RMD_ATTR_VALUE_BEEPER_FREQ_LOW
RMD_ATTR_VALUE_BEEPER_FREQ_MEDIUM
RMD_ATTR_VALUE_BEEPER_FREQ_HIGH

(continued on next page)

The following code example demonstrates how to set the scanner's beeper frequency.

Code Snippet - Set The Beeper Frequency

```
   // ....

   // Create XML string to set the beeper frequency on scanner ID 3 to HIGH.
   NSString *xmlInput = [NSString stringWithFormat:@"
<inArgs><scannerID>%d</scannerID><cmdArgs><arg-xml><attrib_list><attribute><id>%d
</id><datatype>B</datatype><value>%d</value></attribute></attrib_list></arg-xml><
/cmdArgs></inArgs>",3, RMD_ATTR_BEEPER_FREQUENCY,
RMD_ATTR_VALUE_BEEPER_FREQ_HIGH];

   // Command Scanner ID 3 to set beeper frequency to HIGH.
   SBT_RESULT result = [apiInstance sbtExecuteCommand:SBT_RSM_ATTR_SET
aInXML:xmlInput aOutXML:nil forScanner:3];

   if (result == SBT_RESULT_SUCCESS)
   {
       // TODO: Process successful operation
       NSLog(@"Successfully updated beeper settings for scanner ID %d:
%@",scannerId,xmlResponse);
   }
   else
   {
       // TODO: Process error
       NSLog(@"Failed to update beeper settings from scanner ID %d",scannerId);
   }
```

Beep Control

Command the scanner's beeper to emit sound by using the `sbtExecuteCommand` API. Use command op code `SBT_SET_ACTION` and an attribute value shown in *Table 3-4* below:

Table 3-4 *Beeper Attribute Values*

Description	Attribute Values
One high short beep	RMD_ATTR_VALUE_ACTION_HIGH_SHORT_BEEP_1
Two high short beeps	RMD_ATTR_VALUE_ACTION_HIGH_SHORT_BEEP_2
Three high short beeps	RMD_ATTR_VALUE_ACTION_HIGH_SHORT_BEEP_3
Four high short beeps	RMD_ATTR_VALUE_ACTION_HIGH_SHORT_BEEP_4
Five high short beeps	RMD_ATTR_VALUE_ACTION_HIGH_SHORT_BEEP_5
One low short beep	RMD_ATTR_VALUE_ACTION_LOW_SHORT_BEEP_1
Two low short beeps	RMD_ATTR_VALUE_ACTION_LOW_SHORT_BEEP_2
Three low short beeps	RMD_ATTR_VALUE_ACTION_LOW_SHORT_BEEP_3
Four low short beeps	RMD_ATTR_VALUE_ACTION_LOW_SHORT_BEEP_4
Five low short beeps	RMD_ATTR_VALUE_ACTION_LOW_SHORT_BEEP_5

Table 3-4 *Beeper Attribute Values (Continued)*

Description	Attribute Values
One high long beep	RMD_ATTR_VALUE_ACTION_HIGH_LONG_BEEP_1
Two high long beeps	RMD_ATTR_VALUE_ACTION_HIGH_LONG_BEEP_2
Three high long beeps	RMD_ATTR_VALUE_ACTION_HIGH_LONG_BEEP_3
Four high long beeps	RMD_ATTR_VALUE_ACTION_HIGH_LONG_BEEP_4
Five high long beeps	RMD_ATTR_VALUE_ACTION_HIGH_LONG_BEEP_5
One low long beep	RMD_ATTR_VALUE_ACTION_LOW_LONG_BEEP_1
Two low long beeps	RMD_ATTR_VALUE_ACTION_LOW_LONG_BEEP_2
Three low long beeps	RMD_ATTR_VALUE_ACTION_LOW_LONG_BEEP_3
Four low long beeps	RMD_ATTR_VALUE_ACTION_LOW_LONG_BEEP_4
Five low long beeps	RMD_ATTR_VALUE_ACTION_LOW_LONG_BEEP_5
Fast warble beep	RMD_ATTR_VALUE_ACTION_FAST_WARBLE_BEEP
Slow warble beep	RMD_ATTR_VALUE_ACTION_SLOW_WARBLE_BEEP
High-low beep	RMD_ATTR_VALUE_ACTION_HIGH_LOW_BEEP
Low-high beep	RMD_ATTR_VALUE_ACTION_LOW_HIGH_BEEP
High-low-high beep	RMD_ATTR_VALUE_ACTION_HIGH_LOW_HIGH_BEEP
Low-high-low beep	RMD_ATTR_VALUE_ACTION_LOW_HIGH_LOW_BEEP

The following code example demonstrates how to command the scanner's beeper to emit a high short beep sound.

Code Snippet - Beeper Control

```
   // ....
   @import "RMDAttributes.h"

   / ....

   // Specify the ID of the scanner to control
   int scannerID = 1;

   // Specify the scanner ID and attribute value of the beeper command in the XML
   NSString *inXML = [NSString 
stringWithFormat:@"<inArgs><scannerID>%d</scannerID><cmdArgs><arg-int>%d</arg-int
></cmdArgs></inArgs>", scannerID, RMD_ATTR_VALUE_ACTION_HIGH_SHORT_BEEP_1];

   // Issue the beep command
   [scannerSdkApi sbtExecuteCommand:SBT_SET_ACTION aInXML:inXML aOutXML:nil 
forScanner:scannerID];
```

LED

Control the LED

Control the scanner's LED by using the `sbtExecuteCommand` API. Use command op code `SBT_SET_ACTION` and an attribute value shown in *Table 3-5* below.

Table 3-5 *LED Attribute Values*

Description	Attribute Value
Turn Green LED off	`RMD_ATTR_VALUE_ACTION_LED_GREEN_OFF`
Turn Green LED on	`RMD_ATTR_VALUE_ACTION_LED_GREEN_ON`
Turn Yellow LED on	`RMD_ATTR_VALUE_ACTION_LED_YELLOW_ON`
Turn Yellow LED off	`RMD_ATTR_VALUE_ACTION_LED_YELLOW_OFF`
Turn Red LED on	`RMD_ATTR_VALUE_ACTION_LED_RED_ON`
Turn Red LED off	`RMD_ATTR_VALUE_ACTION_LED_RED_OFF`

The following code example demonstrates how to command the scanner's LED to illuminate green.

Code Snippet - LED Control

```
   // ....
   #import "RMDAttributes.h"

   // ....

   // Specify the ID of the scanner to control
   int scannerID = 1;

   // Specify the scanner ID and attribute value the LED command in the XML
   NSString *inXML = [NSString
stringWithFormat:@"<inArgs><scannerID>%d</scannerID><cmdArgs><arg-int>%d</arg-int
></cmdArgs></inArgs>", scannerID, RMD_ATTR_VALUE_ACTION_LED_GREEN_ON];

   // Issue the LED command
   [scannerSdkApi sbtExecuteCommand:SBT_SET_ACTION aInXML:inXML aOutXML:nil
forScanner:scannerID];
```

Scan Capability

Enable Scanning

The scan enable command (SBT_DEVICE_SCAN_ENABLE) is used to enable the device's scanning capability.

The following code example demonstrates how to enable the scan capability.

Code Snippet - Enable Scanning Capability

```
   // ....

   // Create XML string to enable scanning on scanner ID 3
   NSString *xmlInput = [NSString stringWithFormat:@"
<inArgs><scannerID>%d</scannerID></inArgs>",scannerId];

   // Command scanner ID 3 to enable scanning
   SBT_RESULT result = [apiInstance sbtExecuteCommand:SBT_DEVICE_SCAN_ENABLE
aInXML:xmlInput aOutXML:nil forScanner:3];

   if (result == SBT_RESULT_SUCCESS)
   {
       // TODO: Process successful operation
       NSLog(@"Successfully enabled scanning on scanner ID %d",scannerId);
   }
   else
   {
       // TODO: Process error
       NSLog(@"Failed to enable scanning on scanner ID %d",scannerId);
   }
```

Disable Scanning

The scan disable command (SBT_DEVICE_SCAN_DISABLE) is used to disable the device's scanning capability.

The following code example demonstrates how to disable the scan capability.

Code Snippet - Disable Scanning Capability

```
   // ....

   // Create XML string to disable scanning on scanner ID 3
   NSString *xmlInput = [NSString stringWithFormat:@"
<inArgs><scannerID>%d</scannerID></inArgs>",scannerId];

   // Command scanner ID 3 to disable scanning
   SBT_RESULT result = [apiInstance sbtExecuteCommand:SBT_DEVICE_SCAN_DISABLE
aInXML:xmlInput aOutXML:nil forScanner:3];

   if (result == SBT_RESULT_SUCCESS)
   {
       // TODO: Process successful operation
       NSLog(@"Successfully disabled scanning on scanner ID %d",scannerId);
   }
   else
   {
       // TODO: Process error
       NSLog(@"Failed to disable scanning on scanner ID %d",scannerId);
   }
```

Symbologies

Get Supported Symbology IDs

The get all symbologies command (`SBT_RSM_ATTR_GETALL`) is used to retrieve the scanner's support symbologies. The supported symbology attribute values are returned from the scanner as an XML formatted string. *Table 3-6* includes valid symbology attribute values.

Table 3-6 *Symbology Attribute Values*

Description	Attribute Values
UPC-A	`RMD_ATTR_SYM_UPC_A`
UPC-E	`RMD_ATTR_SYM_UPC_E`
UPC-E1	`RMD_ATTR_SYM_UPC_E_1`
EAN-8/JAN8	`RMD_ATTR_SYM_EAN_8_JAN_8`
EAN-13/JAN13	`RMD_ATTR_SYM_EAN_13_JAN_13`
Bookland EAN	`RMD_ATTR_SYM_BOOKLAND_EAN`
Code 128	`RMD_ATTR_SYM_CODE_128`
UCC/EAN-128	`RMD_ATTR_SYM_UCC_EAN_128`
Code 39	`RMD_ATTR_SYM_CODE_39`
Code 93	`RMD_ATTR_SYM_CODE_93`
Code 11	`RMD_ATTR_SYM_CODE_11`
Interleaved 2 of 5	`RMD_ATTR_SYM_INTERLEAVED_2_OF_5`
Discrete 2 of 5	`RMD_ATTR_SYM_DISCRETE_2_OF_5`
Chinese 2 of 5	`RMD_ATTR_SYM_CHINESE_2_OF_5`
Codabar	`RMD_ATTR_SYM_CODABAR`
MSI	`RMD_ATTR_SYM_MSI`
Code 32	`RMD_ATTR_SYM_CODE_32`
Data Matrix	`RMD_ATTR_SYM_DATAMATRIXQR`
PDF	`RMD_ATTR_SYM_PDF`

(continued on next page)

The following code example demonstrates how to get all supported symbologies.

Code Snippet - Retrieve Supported Symbologies

```
   // ....

   // Create XML string to request supported symbologies of scanner ID 3
   NSString *xmlInput = [NSString
stringWithFormat:@"<inArgs><scannerID>%d</scannerID></inArgs>",3];

   NSMutableString *xmlResponse = [[NSMutableString] alloc] init];

   // Retrieve supported symbologies of scanner device that has an ID = 3.
   SBT_RESULT result = [apiInstance sbtExecuteCommand:SBT_RSM_ATTR_GETALL
aInXML:xmlInput aOutXML:&xmlResponse forScanner:3];

   if (result == SBT_RESULT_SUCCESS)
   {
       // TODO: Process xml string containing supported symbologies
       NSLog(@"Supported symbologies from scanner ID %d:
%@",scannerId,xmlResponse);
   }
   else
   {
       // TODO: Process error
       NSLog(@"Failed to retrieve supported symbologies from scanner ID
%d",scannerId);
   }
```

Get Symbology Values

The get attribute command (`SBT_RSM_ATTR_GET`) is used to retrieve the value of a symbology. The symbology value is returned from the scanner as an XML formatted string.

The following code example demonstrates how to retrieve the values of symbologies.

Code Snippet - Retrieve Symbology Value

```
   // ....

   // Create XML string to request value of supported symbology ID's 1 and 8 for
scanner ID 3
   NSString *xmlInput = [NSString
stringWithFormat:@"<inArgs><scannerID>%d</scannerID><cmdArgs><arg-xml><attrib_lis
t>1,8</attrib_list></arg-xml></cmdArgs></inArgs>",3];

   NSMutableString *xmlResponse = [[NSMutableString] alloc] init];

   // Retrieved values of symbologies for scanner ID = 3.
   SBT_RESULT result = [apiInstance sbtExecuteCommand:SBT_RSM_ATTR_GET
aInXML:xmlInput aOutXML:&xmlResponse forScanner:3];

   if (result == SBT_RESULT_SUCCESS)
   {
       // TODO: Process xml string containing symbology values
       NSLog(@"Supported symbology values from scanner ID %d:
%@",scannerId,xmlResponse);
   }
   else
   {
       // TODO: Process error
      NSLog(@"Failed to retrieve symbology values from scanner ID %d",scannerId);
   }
```

Set Symbology Values

The set attribute command (`SBT_RSM_ATTR_SET`) is used to set the value of a symbology. The symbology value is returned from the scanner as an XML formatted string.

The following code example demonstrates how to set the value of symbology.

Code Snippet - Set Symbology Value

```
   // ....

   int scannerId = 3;
   int symbologyId = 8;

   // Create XML string to set value of symbology ID 8 for scanner ID 3 to "True"
   NSString *xmlInput = [NSString stringWithFormat:@"
<inArgs><scannerID>%d</scannerID><cmdArgs><arg-xml><attrib_list><attribute><id>%d
</id><datatype>F</datatype><value>%@</value></attribute></attrib_list></arg-xml><
/cmdArgs></inArgs>",scannerId,symbologyId,"True"];

   // Set value of symbology ID 8 for scanner ID = 3 to "True"
   SBT_RESULT result = [apiInstance sbtExecuteCommand:SBT_RSM_ATTR_SET
aInXML:xmlInput aOutXML:nil forScanner:3];

   if (result == SBT_RESULT_SUCCESS)
   {
       // TODO: Process successful operation
       NSLog(@"Successfully updated symbology value for scanner ID %d:
%@",scannerId);
   }
   else
   {
       // TODO: Process error
       NSLog(@"Failed to set symbology value for scanner ID %d",scannerId);
   }
```

Trigger

Pull Trigger

The pull trigger command (SBT_DEVICE_PULL_TRIGGER) is used to pull the device's trigger. The following code example demonstrates how to command the scanner's trigger to be pulled.

Code Snippet - Pull Trigger

```
   // ....

   // Create XML string to command scanner ID 3 to pull the trigger
   NSString *xmlInput = [NSString stringWithFormat:@"
<inArgs><scannerID>%d</scannerID></inArgs>",scannerId];

   // Command scanner ID 3 to pull the trigger
   SBT_RESULT result = [apiInstance sbtExecuteCommand:SBT_DEVICE_PULL_TRIGGER
aInXML:xmlInput aOutXML:nil forScanner:3];

   if (result == SBT_RESULT_SUCCESS)
   {
       // TODO: Process successful operation
       NSLog(@"Successfully pulled trigger on scanner ID %d: %@",scannerId);
   }
   else
   {
       // TODO: Process error
       NSLog(@"Failed to pull trigger on scanner ID %d",scannerId);
   }
```

Release Trigger

The release trigger command (SBT_DEVICE_RELEASE_TRIGGER) is used to release the device's trigger. The following code example demonstrates how to command the scanner's trigger to be released.

Code Snippet - Release Trigger

```
   // ....

   // Create XML string to command scanner ID 3 to release the trigger
   NSString *xmlInput = [NSString stringWithFormat:@"
<inArgs><scannerID>%d</scannerID></inArgs>",scannerId];

   // Command scanner ID 3 to pull the trigger
   SBT_RESULT result = [apiInstance sbtExecuteCommand:SBT_DEVICE_RELEASE_TRIGGER
aInXML:xmlInput aOutXML:nil forScanner:3];

   if (result == SBT_RESULT_SUCCESS)
   {
       // TODO: Process successful operation
       NSLog(@"Successfully released trigger on scanner ID %d: %@",scannerId);
   }
   else
   {
       // TODO: Process error
       NSLog(@"Failed to release trigger on scanner ID %d",scannerId);
   }
```

AIM

AIM On

The aim on command (`SBT_DEVICE_AIM_ON`) is used to AIM on. The following code example demonstrates how to command the scanner's AIM function should be ON.

Code Snippet - AIM On

```
   // ....

   // Create XML string to command scanner ID 3 to AIM on
    NSString *xmlInput = [NSString stringWithFormat:@"
   <inArgs><scannerID>%d</scannerID></inArgs>",scannerId];

   // Command scanner ID 3 to pull the trigger
   SBT_RESULT result = [apiInstance sbtExecuteCommand:SBT_DEVICE_AIM_ON
aInXML:xmlInput aOutXML:nil forScanner:3];

   if (result == SBT_RESULT_SUCCESS)
   {
   // TODO: Process successful operation
   NSLog(@"Successfully Aim ON on scanner ID %d: %@",scannerId);
   }
   else
   {
   // TODO: Process error
   NSLog(@"Failed to Aim ON on scanner ID %d",scannerId);
   }
```

Aim Off

The aim off command (`SBT_DEVICE_AIM_OFF`) is used to AIM on. The following code example demonstrates how to command the scanner's trigger to be released.

Code Snippet - Aim Off

```
   // ....

   // Create XML string to command scanner ID 3 to AIM off
    NSString *xmlInput = [NSString stringWithFormat:@"
   <inArgs><scannerID>%d</scannerID></inArgs>",scannerId];

   // Command scanner ID 3 to pull the trigger
   SBT_RESULT result = [apiInstance sbtExecuteCommand:SBT_DEVICE_AIM_OFF
aInXML:xmlInput aOutXML:nil forScanner:3];

   if (result == SBT_RESULT_SUCCESS)
   {
   // TODO: Process successful operation
   NSLog(@"Successfully AIM off on scanner ID %d: %@",scannerId);
   }
   else
   {
   // TODO: Process error
   NSLog(@"Failed to AIM off on scanner ID %d",scannerId);
   }
```

Page Motor Function

The vibration feedback command (SBT_DEVICE_VIBRATION_FEEDBACK) is used to activate the page motor. The following code example demonstrates how to command the scanner's page motor to be activated.

Code Snippet - Page Motor Function

```
   // ....

   // Create XML string to command scanner ID 3 to activate page motor.
    NSString *xmlInput = [NSString stringWithFormat:@"
   <inArgs><scannerID>%d</scannerID></inArgs>",scannerId];

   // Command scanner ID 3 to pull the page motor
   SBT_RESULT result = [apiInstance
sbtExecuteCommand:SBT_DEVICE_VIBRATION_FEEDBACK aInXML:xmlInput aOutXML:nil
forScanner:3];

   if (result == SBT_RESULT_SUCCESS)
   {
   // TODO: Process successful operation
   NSLog(@"Successfully activated page motor scanner ID %d: %@",scannerId);
   }
   else
   {
   // TODO: Process error
   NSLog(@"Failed to activate page motor on scanner ID %d",scannerId);
   }
```

Update Firmware with DAT File

The firmware update command (SBT_UPDATE_FIRMWARE) is used to update firmware by DAT file. The following code example demonstrates how to command the scanner to update firmware by DAT file.

Code Snippet - Update Firmware with Dat File

```
   // ....

   // Create XML string to command scanner ID 3 to update firmware
   //selectedFWFilePath = absolute path of the DAT file
    NSString *in_xml = [NSString
stringWithFormat:@"<inArgs><scannerID>%d</scannerID><cmdArgs><arg-string>%@</arg-
string></cmdArgs></inArgs>", m_ScannerID, selectedFWFilePath];

   // Command scanner ID 3 to update firmware
   SBT_RESULT result = [apiInstance sbtExecuteCommand:SBT_UPDATE_FIRMWARE
aInXML:xmlInput aOutXML:nil forScanner:3];

   if (result == SBT_RESULT_SUCCESS)
   {
   // TODO: Process successful operation
   NSLog(@"Successfully pulled updated firmware on scanner ID %d: %@",scannerId);
   }
   else
   {
   // TODO: Process error
   NSLog(@"Failed to update firmware on scanner ID %d",scannerId);
   }
```

Update Firmware with PLUG-in File

The firmware update command (SBT_UPDATE_FIRMWARE_FROM_PLUGIN) is used to update firmware by plug-in file. The following code example demonstrates how to command the scanner to update firmware by plug-in file.

Code Snippet - Update Firmware with Plug-in File

```
   // ....

   // Create XML string to command scanner ID 3 to update firmware by plugin
   //selectedFWFilePath = absolute path of the plugin file
    NSString *in_xml = [NSString 
stringWithFormat:@"<inArgs><scannerID>%d</scannerID><cmdArgs><arg-string>%@</arg-
string></cmdArgs></inArgs>", m_ScannerID, selectedFWFilePath];

   // Command scanner ID 3 to update firmware
   SBT_RESULT result = [apiInstance sbtExecuteCommand:
SBT_UPDATE_FIRMWARE_FROM_PLUGIN aInXML:xmlInput aOutXML:nil forScanner:3];

   if (result == SBT_RESULT_SUCCESS)
   {
   // TODO: Process successful operation
   NSLog(@"Successfully updated firmware on scanner ID %d: %@",scannerId);
   }
   else
   {
   // TODO: Process error
   NSLog(@"Failed to update firmware on scanner ID %d",scannerId);
```

Abort Firmware Update

The abort firmware update command (SBT_DEVICE_ABORT_UPDATE_FIRMWARE) is used to abort ongoing firmware update.

Code Snippet - Update Firmware Update

```
   // ....
   // Create XML string to command scanner ID 3 to update abort fw update
   //
   NSString *in_xml = [NSString 
stringWithFormat:@"<inArgs><scannerID>%d</scannerID></inArgs>", m_ScannerID];

       SBT_RESULT res = [[zt_ScannerAppEngine sharedAppEngine] 
executeCommand:SBT_DEVICE_ABORT_UPDATE_FIRMWARE aInXML:in_xml aOutXML:nil 
forScanner:m_ScannerID];if (result == SBT_RESULT_SUCCESS)
   {
   // TODO: Process successful operation
   NSLog(@"Successfully aborted firmware on scanner ID %d: %@",scannerId);
   }
   else
   {
   // TODO: Process error
   NSLog(@"Failed to abort firmware on scanner ID %d",scannerId);
```

Chapter 4 ZEBRA SCANNER SDK for iOS API

Introduction

This chapter defines the API that can be used by external applications to connect remote scanners to a specific iOS device and control connected scanners.

Application Programming Interface Definition

This section describes constants, types, functions, and notifications which are used in the Zebra Scanner SDK for iOS API.

Constants

Result Codes

These constants are defined to represent result codes that can be returned by SDK API functions.

Table 4-1 *Constants - Result Codes*

Result Code	Description	Value
`SBT_RESULT_SUCCESS`	A specific API function has completed successfully	0
`SBT_RESULT_FAILURE`	A specific API function has completed unsuccessfully.	1
`SBT_RESULT_SCANNER_NOT_AVAILABLE`	A specific API function has completed unsuccessfully because a specified scanner was not available.	2
`SBT_RESULT_SCANNER_NOT_ACTIVE`	A specific API function has completed unsuccessfully because a specified scanner was not active.	3
`SBT_RESULT_INVALID_PARAMS`	A specific API function has completed unsuccessfully due to invalid input and/or output parameters.	4
`SBT_RESULT_RESPONSE_TIMEOUT`	A specific API function has completed unsuccessfully due to expiration of a response timeout during communication with a specific scanner.	5
`SBT_RESULT_OPCODE_NOT_SUPPORTED`	A specific API function has completed unsuccessfully due to unsupported opcode.	6
`SBT_RESULT_SCANNER_NO_SUPPORT`	A specific API function and/or a corresponding SSI command are not supported by a specific model of scanner.	7

Operating Modes

These constants are defined to represent operating modes of the Zebra Scanner SDK for iOS.

Table 4-2 *Constants - Operating Modes*

Operating Mode	Description	Value
`SBT_OPMODE_MFI`	The SDK is able to communicate with scanners in "iOS BT MFi" mode only.	1
`SBT_OPMODE_BTLE`	The SDK is able to communicate with scanners in "iOS BT LE" mode only.	2
`SBT_OPMODE_ALL`	The SDK is able to communicate with scanners in "iOS BT MFi" mode and with scanners in "iOS BT LE" mode.	3

Scanner Modes

These constants are defined to represent communication modes of scanners.

Table 4-3 *Constants - Scanner Modes*

Scanner Mode	Description	Value
`SBT_CONNTYPE_INVALID`	The SDK is unable to determine communication mode of a specific scanner.	0
`SBT_CONNTYPE_MFI`	A specific scanner is in "iOS BT MFi" mode.	1
`SBT_CONNTYPE_BTLE`	A specific scanner is in "iOS BT LE" mode.	2

Notifications

These constants are defined to represent notifications provided by Zebra Scanner SDK for iOS.

Table 4-4 *Constants - Notifications*

Notification	Description	Value
`SBT_EVENT_BARCODE`	"Barcode event" notification (reception of a specific bar code of specific type from a specific active scanner).	1
`SBT_EVENT_IMAGE`	"Image event" notification (triggered when an imaging scanner captures images in image mode).	2
`SBT_EVENT_VIDEO`	"Video event" notification (triggered when an imaging scanner captures video in video mode).	4
`SBT_EVENT_SCANNER_APPEARANCE`	"Device Arrival" notification (appearance of an available scanner).	8
`SBT_EVENT_SCANNER_DISAPPEARANCE`	"Device Disappeared" notification (disappearance of an available scanner).	16
`SBT_EVENT_SESSION_ESTABLISHMENT`	"Session Established" notification (appearance of a specific active scanner).	32
`SBT_EVENT_SESSION_TERMINATION`	"Session Terminated" notification (disappearance of an active scanner).	64
`SBT_EVENT_RAW_DATA`	"Raw Data Received" notification (reception of raw data from a specific active scanner in "raw data pipe" mode).	128

Opcodes

These constants are defined to represent opcodes of methods supported by "Execute Command" API function.

Table 4-5 *Constants - Opcodes*

Opcode	Description	Value
`SBT_DEVICE_PULL_TRIGGER`	Opcode of `DEVICE_PULL_TRIGGER` method of the "Execute Command" API.	2011
`SBT_DEVICE_RELEASE_TRIGGER`	Opcode of `DEVICE_RELEASE_TRIGGER` method of the "Execute Command" API.	2012
`SBT_DEVICE_CAPTURE_IMAGE`	Opcode of `DEVICE_CAPTURE_IMAGE` method of the "Execute Command" API.	3000
`SBT_DEVICE_CAPTURE_VIDEO`	Opcode of `DEVICE_CAPTURE_VIDEO` method of the "Execute Command" API.	4000
`SBT_DEVICE_CAPTURE_BARCODE`	Opcode of `DEVICE_CAPTURE_BARCODE` method of the "Execute Command" API.	3500
`SBT_DEVICE_SCAN_ENABLE`	Opcode of `SCAN_ENABLE` method of the "Execute Command" API.	2014
`SBT_DEVICE_SCAN_DISABLE`	Opcode of `SCAN_DISABLE` method of the "Execute Command" API.	2013
`SBT_SET_ACTION`	Opcode of `SET_ACTION` method of the "Execute Command" API.	6000
`SBT_RSM_ATTR_GETALL`	Opcode of `ATTR_GETALL` method of the "Execute Command" API.	5000
`SBT_RSM_ATTR_GET`	Opcode of `ATTR_GET` method of the "Execute Command" API.	5001
`SBT_RSM_ATTR_GET_OFFSET`	Opcode of `ATTR_GET_OFFSET` method of the "Execute Command" API.	5003
`SBT_RSM_ATTR_SET`	Opcode of `ATTR_SET` method of the "Execute Command" API.	5004
`SBT_RSM_ATTR_STORE`	Opcode of `ATTR_STORE` method of the "Execute Command" API.	5005

Scanner Models

These constants are defined to represent models of scanners supported by Zebra Scanner SDK for iOS.

Table 4-6 *Constants - Scanner Models*

Scanner Model	Description	Value
SBT_DEVMODEL_INVALID	The model either unknown, not recognized or not supported.	0
SBT_DEVMODEL_SSI_RFD8500	RFD8500 in SSI Imager mode.	1
SBT_DEVMODEL_SSI_CS4070	CS4070 scanner.	2
SBT_DEVMODEL_SSI_GENERIC	Generic SSI imager.	3
SBT_DEVMODEL_RFID_RFD8500	RFD8500 in RFID reader mode.	4

LED Codes

These constants are defined to represent LED codes supported by "LED Control" API function.

Table 4-7 *Constants - LED Codes*

LED Code	Description	Value
SBT_LEDCODE_RED	Red LED.	0
SBT_LEDCODE_GREEN	Green LED.	1
SBT_LEDCODE_YELLOW	Yellow LED.	2
SBT_LEDCODE_AMBER	Amber LED.	3
SBT_LEDCODE_BLUE	Blue LED.	4

Beep Codes

These constants are defined to represent beep codes supported by "Beep Control" API function.

Table 4-8 *Constants - Beep Codes*

Beep Code	Description	Value
SBT_BEEPCODE_SHORT_HIGH_1	One high short beep.	0
SBT_BEEPCODE_SHORT_HIGH_2	Two high short beeps.	1
SBT_BEEPCODE_SHORT_HIGH_3	Three high short beeps.	2
SBT_BEEPCODE_SHORT_HIGH_4	Four high short beeps.	3
SBT_BEEPCODE_SHORT_HIGH_5	Five high short beeps.	4
SBT_BEEPCODE_SHORT_LOW_1	One low short beep.	5
SBT_BEEPCODE_SHORT_LOW_2	Two low short beeps.	6
SBT_BEEPCODE_SHORT_LOW_3	Three low short beeps.	7
SBT_BEEPCODE_SHORT_LOW_4	Four low short beeps.	8

Table 4-8 *Constants - Beep Codes (Continued)*

Beep Code	Description	Value
`SBT_BEEPCODE_SHORT_LOW_5`	Five low short beeps.	9
`SBT_BEEPCODE_LONG_HIGH_1`	One high long beep.	10
`SBT_BEEPCODE_LONG_HIGH_2`	Two high long beeps.	11
`SBT_BEEPCODE_LONG_HIGH_3`	Three high long beeps.	12
`SBT_BEEPCODE_LONG_HIGH_4`	Four high long beeps.	13
`SBT_BEEPCODE_LONG_HIGH_5`	Five high long beeps.	14
`SBT_BEEPCODE_LONG_LOW_1`	One low long beep.	15
`SBT_BEEPCODE_LONG_LOW_2`	Two low long beeps.	16
`SBT_BEEPCODE_LONG_LOW_3`	Three low long beeps.	17
`SBT_BEEPCODE_LONG_LOW_4`	Four low long beeps.	18
`SBT_BEEPCODE_LONG_LOW_5`	Five low long beeps.	19
`SBT_BEEPCODE_FAST_WARBLE`	Fast warble beep.	20
`SBT_BEEPCODE_SLOW_WARBLE`	Slow warble beep.	21
`SBT_BEEPCODE_MIX1_HIGH_LOW`	High-low mix beep.	22
`SBT_BEEPCODE_MIX2_LOW_HIGH`	Low-high mix beep.	23
`SBT_BEEPCODE_MIX3_HIGH_LOW_HIGH`	High-low-high mix beep.	24
`SBT_BEEPCODE_MIX4_LOW_HIGH_LOW`	Low-high-low mix beep.	25

Firmware Update Result Codes

Firmware Update Result Code	Description	Value
SBT_FW_UPDATE_RESULT_SUCCESS	Firmware update API call succeeded.	0
SBT_FW_UPDATE_RESULT_FAILURE	Firmware update API call failed.	1

STC Bar Code Types

STC Bar Code Type	Description	Value
BARCODE_TYPE_LEGACY	Legacy bar code type.	0
BARCODE_TYPE_STC	STC 2.0 bar code type.	1

STC Communication Protocol

STC Communication Protocol	Description	Value
STC_SSI_MFI	SSI over MFI.	0
STC_SSI_BLE	SSI over Bluetooth low energy.	1
SBT_SSI_HID	SSI over HID.	2
NO_COM_PROTOCOL	No communication protocol.	3

Set Default Status

Set Default Status	Description	Value
SETDEFAULT_YES	Set default yes.	0
SETDEFAULT_NO	Set default no.	1

Types

SBT_RESULT

Type of return value of Zebra Scanner SDK for iOS API functions.

```
typedef enum {
    SBT_RESULT_SUCCESS                      = 0x00,
    SBT_RESULT_FAILURE                      = 0x01,
    SBT_RESULT_SCANNER_NOT_AVAILABLE        = 0x02,
    SBT_RESULT_SCANNER_NOT_ACTIVE           = 0x03,
    SBT_RESULT_INVALID_PARAMS               = 0x04,
    SBT_RESULT_RESPONSE_TIMEOUT             = 0x05,
    SBT_RESULT_OPCODE_NOT_SUPPORTED         = 0x06,
    SBT_RESULT_SCANNER_NO_SUPPORT           = 0x07
} SBT_RESULT;
```

SbtScannerInfo

Used to represent a specific scanner.

```
@interface SbtScannerInfo : NSObject
{
    int m_ScannerID;
    int m_ConnectionType;
    BOOL m_AutoCommunicationSessionReestablishment;
    BOOL m_Active;
    BOOL m_Available;
    NSString *m_ScannerName;
    int m_ScannerModel;
}
```

(continued on next page)

```
- (id) init;
- (void) dealloc;
- (void) setScannerID:(int)scannerID;
- (void) setConnectionType:(int)connectionType;
- (void) setAutoCommunicationSessionReestablishment:(BOOL)enable;
- (void) setActive:(BOOL)active;
- (void) setAvailable:(BOOL)available;
- (void) setScannerName:(NSString*)scannerName;
- (void) setScannerModel:(int)scannerModel;

- (int) getScannerID;
- (int) getConnectionType;
- (BOOL) getAutoCommunicationSessionReestablishment;
- (BOOL) isActive;
- (BOOL) isAvailable;
- (NSString*) getScannerName;
- (int) getScannerModel;

@end
```

Table 4-9 *SbtScannerInfo Variable Descriptions*

Variable Name	Description
m_ScannerID	Unique identifier of a specific scanner assigned by SDK.
m_ConnectionType	Communication mode of a specific scanner.
m_AutoCommunicationSessionReestablishment	State of "Automatic communication session reestablishment" option.
m_Active	State of a specific scanner (i.e., active scanner is a scanner with which a communication session was already established).
m_ScannerName	Name of a specific scanner.
m_ScannerModel	Model code of a specific scanner.

ISbtSdkApi

Objective C protocol which defines SDK API functions.

```
@protocol ISbtSdkApi <NSObject>

- (SBT_RESULT) sbtSetDelegate:(id<ISbtSdkApiDelegate>)delegate;
- (NSString*) sbtGetVersion;
- (SBT_RESULT) sbtSetOperationalMode:(int)operationalMode;
- (SBT_RESULT) sbtSubsribeForEvents:(int)sdkEventsMask;
- (SBT_RESULT) sbtUnsubsribeForEvents:(int)sdkEventsMask;
- (SBT_RESULT) sbtGetAvailableScannersList:(NSMutableArray**)availableScannersList;
- (SBT_RESULT) sbtGetActiveScannersList:(NSMutableArray**)activeScannersList;
- (SBT_RESULT) sbtEstablishCommunicationSession:(int)scannerID;
- (SBT_RESULT) sbtTerminateCommunicationSession:(int)scannerID;
- (SBT_RESULT) sbtEnableAvailableScannersDetection:(BOOL)enable;
- (SBT_RESULT) sbtEnableAutomaticSessionReestablishment:(BOOL)enable
forScanner:(int)scannerID;
```

(continued on next page)

```
- (SBT_RESULT) sbtExecuteCommand:(int)opCode aInXML:(NSString*)inXML
aOutXML:(NSMutableString**)outXML forScanner:(int)scannerID;
- (SBT_RESULT) sbtLedControl:(BOOL)enable aLedCode:(int)ledCode forScanner:(int)scannerID;
- (SBT_RESULT) sbtBeepControl:(int)beepCode forScanner:(int)scannerID;
- (void) sbtSetBTAddress:(NSString*)btAdd;
- (UIImage*) sbtGetPairingBarcode:(BARCODE_TYPE)barcodeType
withComProtocol:(STC_COM_PROTOCOL)comProtocol
withSetDefaultStatus:(SETDEFAULT_STATUS)setDefaultsStatus
withBTAddress:(NSString*)btAddress withImageFrame:(CGRect)imageFrame;

@end
```

See *Functions on page 4-10* for the descriptions of API functions.

SbtSdkFactory

Used to create and access a single shared instance of an API object. API object implements `ISbtSdkApi` protocol and is used to perform specific API calls.

```
@interface SbtSdkFactory : NSObject

+(id<ISbtSdkApi>) createSbtSdkApiInstance;

@end
```

Table 4-10 *SbtScannerInfo Variable Descriptions*

Name	Description
`+(id<ISbtSdkApi>)createSbtSdkApiInstance`	Class method, returns a single shared instance of an object which conforms to `ISbtSdkApi` protocol and should be used to perform specific API calls.

ISbtSdkApiDelegate

Objective C protocol which defines the SDK callbacks interface. Registration of a specific object which conforms to the `ISbtSdkApiDelegate` protocol is required to receive particular from the SDK.

```
@protocol ISbtSdkApiDelegate <NSObject>

- (void) sbtEventScannerAppeared:(SbtScannerInfo*)availableScanner;
- (void) sbtEventScannerDisappeared:(int)scannerID;
- (void) sbtEventCommunicationSessionEstablished:(SbtScannerInfo*)activeScanner;
- (void) sbtEventCommunicationSessionTerminated:(int)scannerID;
- (void) sbtEventBarcode:(NSString*)barcodeData barcodeType:(int)barcodeType
fromScanner:(int)scannerID;
- (void) sbtEventBarcodeData:(NSData *)barcodeData barcodeType:(int)barcodeType
fromScanner:(int)scannerID
- (void) sbtEventImage:(NSData*)imageData fromScanner:(int)scannerID;
- (void) sbtEventVideo:(NSData*)videoFrame fromScanner:(int)scannerID;

@end
```

See *Notifications on page 4-20* for descriptions of SDK callbacks.

SBT_FW_UPDATE_RESULT

Type of return value of Firmware update function.

```
typedef enum {
    SBT_FW_UPDATE_RESULT_SUCCESS                     = 0x00,
    SBT_FW_UPDATE_RESULT_FAILURE                     = 0x01,
} SBT_FW_UPDATE_RESULT;
```

STC_COM_PROTOCOL

Communication protocol.

```
typedef enum {
    STC_SSI_MFI                    = 0x00,
    STC_SSI_BLE                    = 0x01,
    SBT_SSI_HID                    = 0x02,
    NO_COM_PROTOCOL                = 0X03
} STC_COM_PROTOCOL
```

SETDEFAULT_STATUS

Set default or not.

```
typedef enum {
    SETDEFAULT_YES                    = 0x00,
    SETDEFAULT_NO                     = 0x01,
} SETDEFAULT_STATUS;
```

BARCODE_TYPE

Type of the bar code.

```
typedef enum {
    BARCODE_TYPE_LEGACY                   = 0x00,
    BARCODE_TYPE_STC                    = 0x01,
} BARCODE_TYPE;
```

Functions

API functions are defined by `ISbtSdkApi` Objective C protocol. A specific object which implements the protocol is accessible via `createSbtSdkApiInstance` method of `SbtSdkFactory` class. See *ISbtSdkApi on page 4-8* and *SbtSdkFactory on page 4-9* for details.

sbtGetVersion

Description

Returns version of the SDK.

```
- (NSString*) sbtGetVersion;
```

Parameters

None.

Return Values

SDK version as NSString autoreleased object.

sbtSetDelegate

Description

Registers a specific object which conforms to `ISbtSdkApiDelegate` Objective C protocol as a receiver of SDK notifications. Registration of a specific object which conforms to `ISbtSdkApiDelegate` protocol is required to receive notifications from the SDK.

```
- (SBT_RESULT) sbtSetDelegate:(id<ISbtSdkApiDelegate>)delegate;
```

Parameters

`(id<ISbtSdkApiDelegate>)delegate`

[in] An object which conforms to `ISbtSdkApiDelegate` protocol.

Return Values

`SBT_RESULT_SUCCESS`

The receiver of SDK notifications was registered successfully.

sbtSetOperationalMode

Description

Configures operating mode of the SDK.

```
- (SBT_RESULT) sbtSetOperationalMode:(int)operationalMode;
```

Parameters

`(int)operationalMode`

[in] Identifier of requested operating mode.

Return Values

`SBT_RESULT_SUCCESS`

The requested operating mode of the SDK was configured successfully.

`SBT_RESULT_FAILURE`

Invalid parameters.

✓ ***NOTES*** If operating mode of the SDK is not configured intentionally, the SDK remains disabled and is not able to communicate with scanners in either "iOS BT MFi" or "iOS BT LE" mode.

Selection of a new operating mode results in:

- Disconnection of active incompatible scanners (e.g., "iOS BT LE" scanners in SBT_OPMODE_MFI mode), removing available incompatible scanners and providing corresponding notifications if these notifications are enabled.
- Performing discovery of available scanners compatible with requested operating mode (if "Available scanners detection" option is enabled) and providing corresponding notifications if these notifications are enabled.

sbtSubsribeForEvents

Description

Enables providing of notification of requested types.
`- (SBT_RESULT) sbtSubsribeForEvents:(int)sdkEventsMask;`

Parameters

`(int)sdkEventsMask`

[in] Subscription option flags.

Return Values

`SBT_RESULT_SUCCESS`

Subscription was performed successfully.

sbtUnsubsribeForEvents

Description

Disables providing of notification of requested types.
`- (SBT_RESULT) sbtUnsubsribeForEvents:(int)sdkEventsMask;`

Parameters

`(int)sdkEventsMask`

[in] Unsubscription option flags.

Return Values

`SBT_RESULT_SUCCESS`

Unsubscription was performed successfully.

sbtGetAvailableScannersList

Description

Requests the list of available scanners.
`- (SBT_RESULT) sbtGetAvailableScannersList:(NSMutableArray**)availableScannersList;`

Parameters

`(NSMutableArray**)availableScannersList`

[out] Pointer to `NSMutableArray` object which contains `SbtScannerInfo` objects that represents available scanners.

Return Values

`SBT_RESULT_SUCCESS`

Discovery procedure was performed and `availableScannersList` parameter is filled with available scanners.

`SBT_RESULT_INVALID_PARAMS`

Invalid parameter (e.g., nil pointer).

✓ ***NOTES***

- The caller is responsible for allocation, initialization and deallocation of `NSMutableArray` object.
- All objects from received `NSMutableArray` object are removed.
- The SDK performs discovery of available scanners without providing any notifications.
- Received `NSMutableArray` object is filled with `SbtScannerInfo` objects that represents available scanners.

sbtGetActiveScannersList

Description

Requests the list of active scanners.
`- (SBT_RESULT) sbtGetActiveScannersList:(NSMutableArray**)activeScannersList;`

Parameters

`(NSMutableArray**)activeScannersList`

[out] Pointer to `NSMutableArray` object which contains `SbtScannerInfo` objects that represents active scanners.

Return Values

`SBT_RESULT_SUCCESS`

Discovery procedure was performed and `activeScannersList` parameter is filled with available scanners.

`SBT_RESULT_INVALID_PARAMS`

Invalid parameter (e.g., nil pointer).

✓ ***NOTES***

- The caller is responsible for allocation, initialization and deallocation of `NSMutableArray` object.
- All objects from received `NSMutableArray` object are removed.
- Received `NSMutableArray` object is filled with `SbtScannerInfo` objects that represents available scanners.

sbtEstablishCommunicationSession

Description

Requests to establish communication session with a specific available scanner in "SSI" mode.

- (SBT_RESULT) `sbtEstablishCommunicationSession:(int)scannerID`;

Parameters

`(int)scannerID`

[in] Unique identifier of a specific scanner assigned by SDK.

Return Values

`SBT_RESULT_SUCCESS`

The communication session was established successfully.

`SBT_RESULT_FAILURE`

The communication session was not established.

`SBT_RESULT_SCANNER_NO_SUPPORT`

A specific scanner does not support SSI communication mode.

✓ ***NOTE*** "Session Established" notification is provided if this type of notification is enabled.

sbtTerminateCommunicationSession

Description

Requests to terminate communication session with a specific active scanner.

```
- (SBT_RESULT) sbtTerminateCommunicationSession:(int)scannerID;
```

Parameters

`(int)scannerID`

[in] Unique identifier of a specific scanner assigned by SDK.

Return Values

`SBT_RESULT_SUCCESS`

The communication session was terminated successfully.

`SBT_RESULT_FAILURE`

The communication session was not terminated.

✓ ***NOTE*** "Session Termination" notification is provided if this type of notification is enabled.

sbtEnableAvailableScannersDetection

Description

Requests to enable/disable "Available scanners detection" option.

```
- (SBT_RESULT) sbtEnableAvailableScannersDetection:(BOOL)enable;
```

Parameters

`(BOOL)enable`

[in] Whether the option should be enabled or disabled:

YES

Requests to enable "Available scanners detection" option.

NO

Requests to disable "Available scanners detection" option.

Return Values

`SBT_RESULT_SUCCESS`

"Available scanners detection" option was enabled/disabled successfully.

✓ ***NOTES***

- The SDK performs a discovery of available scanners once the option is enabled and provides corresponding notifications if the notifications are enabled.
- If the option is enabled, the SDK detects connection and disconnection of scanners operating in "iOS BT MFi" mode in both foreground and background execution modes of a specific application linked with the SDK and provides corresponding notifications if the notifications are enabled.
- If the option is enabled the SDK detects appearance and disappearance of scanners operating in "iOS BT LE" mode in only foreground execution mode of a specific application linked with the SDK through periodic discovery operation and provides corresponding notifications if the notifications are enabled.

sbtEnableAutomaticSessionReestablishment

Description

Requests to enable/disable "Automatic communication session reestablishment" option for a specific scanner.

```
- (SBT_RESULT) sbtEnableAutomaticSessionReestablishment:(BOOL)enable
forScanner:(int)scannerID;
```

Parameters

`(int)scannerID`

[in] Unique identifier of a specific scanner assigned by SDK.

`(BOOL)enable`

[in] Whether the option should be enabled or disabled:

YES

Requests to enable "Automatic communication session reestablishment" option.

NO

Requests to disable "Automatic communication session reestablishment" option.

Return Values

`SBT_RESULT_SUCCESS`

"Automatic communication session reestablishment" option was enabled/disabled successfully.

`SBT_RESULT_FAILURE`

The specified scanner was not found.

✓ ***NOTES***

- If the option is enabled for a specific scanner the SDK automatically establishes a communication session in "SSI" mode with this scanner once the scanner is recognized as available:
- The scanner can be recognized as available automatically by the SDK if the "Available scanners detection" option is enabled.
- The scanner can be recognized as available during discovery procedure requested by the `sbtGetAvailableScannersList` API.
- "Session Established" notification is provided once the communication session is established if this type of notification is enabled.

sbtExecuteCommand

Description

Provides synchronous execution of a specific method via an opcode. See *Methods of the "Execute Command" API on page 4-23* for description of specific methods.

```
- (SBT_RESULT) sbtExecuteCommand:(int)opCode aInXML:(NSString*)inXML
aOutXML:(NSMutableString**)outXML forScanner:(int)scannerID;
```

Parameters

`(int)opCode`

[in] Method to be executed.

`(NSString*)inXML`

[in] Relevant argument list for the opcode, structured into an XML string.

`(NSMutableString**)outXML`

[out] Results of method execution, structured into an XML string.

`(int)scannerID`

[in] Unique identifier of a specific scanner assigned by SDK.

Return Values

`SBT_RESULT_SUCCESS`

The requested method was executed successfully.

SBT_RESULT_SCANNER_NOT_AVAILABLE

The requested method was not executed because the scanner specified by scannerID parameter was not available.

`SBT_RESULT_SCANNER_NOT_ACTIVE`

The requested method was not executed because the scanner specified by scannerID parameter was not active.

`SBT_RESULT_OPCODE_NOT_SUPPORTED`

The requested method is not supported by SDK.

`SBT_RESULT_INVALID_PARAMS`

The requested method was not executed due to invalid input and/or output parameters.

`SBT_RESULT_RESPONSE_TIMEOUT`

The requested method was completed unsuccessfully due to expiration of a response timeout during communication with a specific scanner

`SBT_RESULT_FAILURE`

The requested method was executed unsuccessfully.

`SBT_RESULT_SCANNER_NO_SUPPORT`

The requested method is not supported by the scanner.

✓ ***NOTES***

- The caller is responsible for allocation, initialization and deallocation of `outXML NSMutableString` object.
- `scannerID` identifier of scanner is the same as scanner ID identifier provided in `inXML` string argument.

sbtLedControl

Description

Perform an action involving the scanner's LEDs. See *LED Codes on page 4-5* for supported LED codes.

```
- (SBT_RESULT) sbtLedControl:(BOOL)enable aLedCode:(int)ledCode
forScanner:(int)scannerID;
```

Deprecation Statement

This API will be removed in the future. Use the `sbtExecuteCommand` function instead to perform an LED action.

Parameters

`(BOOL)enable`

[in] Whether a specified LED is turned on or off.

`(int)ledCode`

[in] Code of a specific LED.

`(int)scannerID`

[in] Unique identifier of a specific scanner assigned by SDK.

Return Values

`SBT_RESULT_SUCCESS`

LED control operation was performed successfully.

`SBT_RESULT_SCANNER_NOT_AVAILABLE`

LED control operation was not performed because the scanner specified by `scannerID` parameter was not available.

`SBT_RESULT_SCANNER_NOT_ACTIVE`

LED control operation was not performed because the scanner specified by `scannerID` parameter was not active.

`SBT_RESULT_INVALID_PARAMS`

LED control operation was not performed due to invalid LED code or LED code that is not supported by the scanner.

`SBT_RESULT_RESPONSE_TIMEOUT`

LED control operation was completed unsuccessfully due to expiration of a response timeout during communication with a specific scanner

`SBT_RESULT_FAILURE`

LED control operation was executed unsuccessfully.

`SBT_RESULT_SCANNER_NO_SUPPORT`

A specific scanner does not support required SSI commands.

✓ ***NOTES***

- The LED control operation is performed through sending `LED_ON` or `LED_OFF` SSI protocol command.
- The LED control operation is supposed to be performed successfully if `CMD_ACK` is received from a scanner.

sbtBeepControl

Description

Perform an action involving the scanner's beeper. See *Beep Codes on page 4-5* for supported beep codes.

```
- (SBT_RESULT) sbtBeepControl:(int)beepCode forScanner:(int)scannerID;
```

Deprecation Statement

This API will be removed in the future. Use the `sbtExecuteCommand` function instead to perform a beeper action.

Parameters

`(int)beepCode`

[in] A specific beep code.

`(int)scannerID`

[in] Unique identifier of a specific scanner assigned by SDK.

Return Values

`SBT_RESULT_SUCCESS`

The beep control operation was performed successfully.

`SBT_RESULT_SCANNER_NOT_AVAILABLE`

The beep control operation was not performed because the scanner specified by `scannerID` parameter was not available.

`SBT_RESULT_SCANNER_NOT_ACTIVE`

The beep control operation was not performed because the scanner specified by `scannerID` parameter was not active.

`SBT_RESULT_INVALID_PARAMS`

The beep control operation was not performed due to invalid beep code or beep code that is not supported by the scanner.

`SBT_RESULT_RESPONSE_TIMEOUT`

The beep control operation was completed unsuccessfully due to expiration of a response timeout during communication with a specific scanner

`SBT_RESULT_FAILURE`

The beep control operation was executed unsuccessfully.

`SBT_RESULT_SCANNER_NO_SUPPORT`

A specific scanner does not support required SSI commands.

NOTES

- The beep control operation is performed through sending BEEP SSI protocol command for action commands involving scanner's beeper.
- The beep control operation is supposed to be performed successfully if `CMD_ACK` is received from a scanner.

sbtSetBTAddress

Description

Set Bluetooth address to generate the STC bar code.

```
- (void) sbtSetBTAddress:(NSString*)btAdd;
```

Parameters

`(NSString*)btAdd`

[in] Bluetooth address.

sbtGetPairingBarcode

Description

Get pairing bar code.

```
- (UIImage*) sbtGetPairingBarcode:(BARCODE_TYPE)barcodeType
withComProtocol:(STC_COM_PROTOCOL)comProtocol
withSetDefaultStatus:(SETDEFAULT_STATUS)setDefaultsStatus
withBTAddress:(NSString*)btAddress withImageFrame:(CGRect)imageFrame;
```

Deprecation Statement

This API will be removed in the future. Use the `sbtExecuteCommand` function instead to perform a beeper action.

Parameters

`(BARCODE_TYPE)barcodeType`

[in] Type of the bar code.

`(STC_COM_PROTOCOL)comProtocol`

[in] Communication protocol.

`(SETDEFAULT_STATUS)setDefaultsStatus`

[in] Set default status.

`(NSString*)btAddress`

[in] Bluetooth address.

`(CGRect)imageFrame`

[in] Bluetooth address.

Return Value

`UIImage`

Image of the generated STC bar code.

Notifications

The SDK callback interface is defined by `ISbtSdkApiDelegate` Objective C protocol. Registration of a specific object which conforms to `ISbtSdkApiDelegate` protocol is required to receive specific notifications from Zebra Scanner SDK for iOS.

sbtEventScannerAppeared

Description

"Device Arrival" notification informs about appearance of a specific available scanner.
`- (void) sbtEventScannerAppeared:(SbtScannerInfo*)availableScanner`

Parameters

`(SbtScannerInfo*)availableScanner`

`SbtScannerInfo` object representing an appeared available scanner.

✓ ***NOTE*** The SDK is responsible for allocation and deallocation of `availableScanner` object. The SDK deallocates the `availableScanner` object immediately after execution of the callback.

sbtEventScannerDisappeared

Description

"Device Disappeared" notification informs about disappearance of a specific available scanner.
`- (void) sbtEventScannerDisappeared:(int)scannerID;`

Parameters

`(int)scannerID`

Unique identifier of a disappeared available scanner assigned by SDK.

sbtEventCommunicationSessionEstablished

Description

"Session Established" notification informs about appearance of a specific active scanner.
`- (void) sbtEventCommunicationSessionEstablished:(SbtScannerInfo*)activeScanner;`

Parameters

`(SbtScannerInfo*)activeScanner`

SbtScannerInfo object representing an appeared active scanner.

✓ ***NOTE*** The SDK is responsible for allocation and deallocation of activeScanner object. The SDK deallocates the activeScanner object immediately after execution of the callback.

sbtEventCommunicationSessionTerminated

Description

"Session Terminated" notification informs about disappearance of a specific active scanner
`- (void) sbtEventCommunicationSessionTerminated:(int)scannerID;`

Parameters

`(int)scannerID`

Unique identifier of a disappeared active scanner assigned by SDK.

sbtEventBarcode (deprecated)

Description

"Barcode Event" notification informs about reception of a specific bar code of a specific type from a specific active scanner.

```
- (void) sbtEventBarcode:(NSString*)barcodeData barcodeType:(int)barcodeType
fromScanner:(int)scannerID;
```

Deprecation Statement

This API will be removed in the future. Use the `sbtEventBarcodeData` function instead (see *sbtEventBarcodeData* below).

Parameters

`(NSString*)barcodeData`

`NSString` object representing raw data of the scanned bar code.

`(int)barcodeType`

Bar code type of the scanned bar code. Values of bar code data types are available in the *Zebra Scanner SDK For Windows Developer Guide* (p/n 72E-149784-XX).

`(int)scannerID`

Unique identifier of a specific active scanner assigned by SDK.

✓ ***NOTE*** The SDK is responsible for allocation and deallocation of `barcodeData` object. The SDK deallocates the `barcodeData` object immediately after execution of the callback.

sbtEventBarcodeData

Description

"Barcode Event" notification informs about reception of a particular barcode of a particular type from a particular active scanner.

```
- (void) sbtEventBarcodeData:(NSData*)barcodeData barcodeType:(int)barcodeType
fromScanner:(int)scannerID;
```

Parameters

`(NSData*)barcodeData`

`NSData` object representing raw data byte of the scanned bar code.

`(int)barcodeType`

Bar code type of the scanned bar code. Values of bar code data types are available in the *Zebra Scanner SDK For Windows Developer Guide* (p/n 72E-149784-XX).

`(int)scannerID`

Unique identifier of a particular active scanner assigned by SDK.

✓ ***NOTE*** - Use this function instead of `sbtEventBarcode`.

sbtEventImage

Description

"Image Event" notification is triggered when an active imaging scanner captures images in image mode.
`- (void) sbtEventImage:(NSData*)imageData fromScanner:(int)scannerID;`

Parameters

`(NSData*)barcodeData`

`NSData` object representing raw data of the received image.

`(int)scannerID`

Unique identifier of a specific active scanner assigned by SDK.

✓ ***NOTE*** The SDK is responsible for allocation and deallocation of `imageData` object. The SDK deallocates the `imageData` object immediately after execution of the callback.

Not implemented.

sbtEventVideo

Description

"Video Event" notification is triggered when an active imaging scanner captures video in video mode.
`(void)sbtEventVideo:(NSData*)videoFrame fromScanner:(int)scannerID;`

Parameters

`(NSData*)videoFrame`

`NSData` object representing raw data of the received video frame.

`(int)scannerID`

Unique identifier of a specific active scanner assigned by SDK.

✓ ***NOTE*** The SDK is responsible for allocation and deallocation of `videoFrame` object. The SDK deallocates the `videoFrame` object immediately after execution of the callback.

Not implemented.

sbtEventFirmwareUpdate

Description

"Firmware update" notification informs about current status during a firmware update.
`(void)sbtEventVideo:(NSData*)videoFrame fromScanner:(int)scannerID;`

Parameters

`(FirmwareUpdateEvent *) event`

FirmwareUpdateEvent object representing current status of firmware update.

✓ ***NOTE*** No need to specifically subscribe to this event.

Methods of the "Execute Command" API

The described methods are invoked through the "Execute Command" API. See *sbtExecuteCommand on page 4-16*t for details. The execution of most of the described methods results in communicating with a specific active scanner via SSI and RMD protocols. Refer to the *Simple Serial Interface Version 2.0 Internal Specification* and RMD protocol specification for details.

DEVICE_PULL_TRIGGER

Pull the trigger of a specified active scanner.

Opcode

```
SBT_DEVICE_PULL_TRIGGER
```

inXML

```
<inArgs>
  <scannerID>1</scannerID>                 ? Specified Scanner ID
</inArgs>
```

outXML

Not used.

✓ ***NOTES***
- Execution of the method results in sending `START_SESSION` SSI protocol command.
- The method is supposed to be executed successfully if `CMD_ACK` is received from a scanner.

DEVICE_RELEASE_TRIGGER

Release the pulled trigger of a specified active scanner.

Opcode

```
SBT_DEVICE_RELEASE_TRIGGER
```

inXML

```
<inArgs>
  <scannerID>1</scannerID>                 ? Specified Scanner ID
</inArgs>
```

outXML

Not used.

✓ ***NOTES***
- Execution of the method results in sending `STOP_SESSION` SSI protocol command.
- The method is supposed to be executed successfully if `CMD_ACK` is received from a scanner.

SCAN_DISABLE

Disable scanning of a specified active scanner.

Opcode

```
SBT_DEVICE_SCAN_DISABLE
```

inXML

```
<inArgs>
  <scannerID>1</scannerID>                    ? Specified Scanner ID
</inArgs>
```

outXML

Not used.

NOTES

- Execution of the method results in sending SCAN_DISABLE SSI protocol command.
- The method is supposed to be executed successfully if CMD_ACK is received from a scanner.

SCAN_ENABLE

Enable scanning on a specified active scanner.

Opcode

```
SBT_DEVICE_SCAN_ENABLE
```

inXML

```
<inArgs>
  <scannerID>1</scannerID>                    ? Specified Scanner ID
</inArgs>
```

outXML

Not used.

NOTES

- Execution of the method results in sending SCAN_ENABLE SSI protocol command.
- The method is supposed to be executed successfully if CMD_ACK is received from a scanner.

DEVICE_CAPTURE_BARCODE

Change a specified active scanner to decode mode.

Opcode

`SBT_DEVICE_CAPTURE_BARCODE`

inXML

```
<inArgs>
  <scannerID>1</scannerID>                 ? Specified Scanner ID
</inArgs>
```

outXML

Not used.

✓ ***NOTES***

- Execution of the method results in sending `IMAGER_MODE` SSI protocol command.
- The method is supposed to be executed successfully if `CMD_ACK` is received from a scanner.

DEVICE_CAPTURE_IMAGE

Change a specified active scanner to snapshot mode.

Opcode

`SBT_DEVICE_CAPTURE_IMAGE`

inXML

```
<inArgs>
  <scannerID>1</scannerID>                 ? Specified Scanner ID
</inArgs>
```

outXML

Not used.

✓ ***NOTES***

- Execution of the method results in sending `IMAGER_MODE` SSI protocol command.
- The method is supposed to be executed successfully if `CMD_ACK` is received from a scanner.

DEVICE_CAPTURE_VIDEO

Change a specified active scanner to video mode.

Opcode

SBT_DEVICE_CAPTURE_VIDEO

inXML

```
<inArgs>
  <scannerID>1</scannerID>                    ? Specified Scanner ID
</inArgs>
```

outXML

Not used.

✓ ***NOTES***

- Execution of the method results in sending IMAGER_MODE SSI protocol command.
- - The method is supposed to be executed successfully if CMD_ACK is received from a scanner.

SET_ACTION

Perform an action involving the scanner's beeper or LEDs. Values for the SET_ACTION method are available in the *Zebra Scanner SDK Attribute Dictionary* (p/n 72E-149786-XX).

Opcode

SBT_SET_ACTION

inXML

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

outXML

Not used.

✓ ***NOTES***

- Execution of the method results in sending ATTRIBUTE_SET RMD protocol command via SSI_MGMT_COMMAND SSI protocol command in order to set a required value of "Beeper/LED" attribute.
- The method is supposed to be executed successfully if ATTRIBUTE_SET RMD protocol command for "Beeper/LED" attribute was sent to a scanner and a corresponding response for the issued ATTRIBUTE_SET RMD command is received via SSI_MGMT_COMMAND SSI packets from a scanner.

ATTR_GETALL

Get all attributes of a specified active scanner. Refer to the *Zebra Scanner SDK Attribute Dictionary* (p/n 72E-149786-XX) for attribute numbers, types and possible values.

Opcode

SBT_RSM_ATTR_GETALL

inXML

```
<inArgs>
  <scannerID>1</scannerID>                 ? Specified Scanner ID
</inArgs>
```

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>
    <response>
      <opcode>5000</opcode>                ? Method response received
      <attrib_list>
        <attribute name="">0</attribute>   ? Attribute Numbers
        <attribute name="">1</attribute>
        <attribute name="">2</attribute>
        <attribute name="">17</attribute>
        <attribute name="">18</attribute>
        <attribute name="">19</attribute>
        <attribute name="">655</attribute>
        <attribute name="">656</attribute>
        <attribute name="">657</attribute>
        <attribute name="">705</attribute>
        <attribute name="">716</attribute>
        <attribute name="">718</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>
```

NOTES

- Execution of the method results in sending `ATTRIBUTE_GETALL` RMD protocol command(s) via `SSI_MGMT_COMMAND` SSI protocol command.
- The method is supposed to be executed successfully if a set of `ATTRIBUTE_GETALL` RMD responses are received via `SSI_MGMT_COMMAND` packets from a scanner and the last response packet indicates that the end of the attribute table was reached.

ATTR_GET

Query the values of attribute(s) of a specified active scanner. Refer to the *Zebra Scanner SDK Attribute Dictionary* (p/n 72E-149786-XX) for attribute numbers, types and possible values.

Opcode

SBT_RSM_ATTR_GET

inXML

```
<inArgs>
  <scannerID>1</scannerID>                           ? Specified Scanner ID
  <cmdArgs>
    <arg-xml>
      <attrib_list>535,20004,1,140,392</attrib_list>  ? Attribute Numbers
    </arg-xml>
  </cmdArgs>
</inArgs>
```

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>
    <response>
      <opcode>5001</opcode>                           ? Method Response Received
      <attrib_list>
        <attribute>
          <id>535</id>                                ? Attribute Number
          <name></name>
          <datatype>S</datatype>                      ? Attribute Data Type
          <permission>R</permission>                  ? Permissions of the Attribute
          <value>27APR07</value>                      ? Attribute Value
        </attribute>
        <attribute>
          <id>20004</id>
          <name></name>
          <datatype>S</datatype>
          <permission>R</permission>
          <value>DS6707X4</value>
        </attribute>
        <attribute>
          <id>1</id>
          <name></name>
          <datatype>F</datatype>
          <permission>RWP</permission>
          <value>True</value>
        </attribute>
        <attribute>
          <id>140</id>
          <name></name>
          <datatype>B</datatype>
          <permission>RWP</permission>
          <value>0</value>
```

(continued on next page)

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

NOTES

- Execution of the method results in sending `ATTRIBUTE_GET` RMD protocol command(s) via `SSI_MGMT_COMMAND` SSI protocol command.
- The method should be executed successfully if all required attributes are requested from a scanner through a set of `ATTRIBUTE_GET` RMD commands and responses for all issued `ATTRIBUTE_GET` RMD commands are received via `SSI_MGMT_COMMAND` SSI packets from a scanner.
- The successful execution of the method does not guarantee that all requested attributes are received from a scanner if some of requested attributes are not supported by a specific scanner.
- The method supports the query of maximum 100 attributes values. If more than 100 attribute values are requested, the "Execute Command" API returns `SBT_RESULT_INVALID_PARAMS` to indicate that input parameters are not valid.

ATTR_GET_OFFSET

Query the value of an attribute of a specified active scanner. The method provides the ability to retrieve string/array attribute values that do not fit within a single RMD protocol packet. It is the responsibility of the application to determine what the starting offset is. Refer to the *Zebra Scanner SDK Attribute Dictionary* (p/n 72E-149786-XX) for attribute numbers, types, and possible values.

Opcode

```
SBT_RSM_ATTR_GET_OFFSET
```

inXML

```
<inArgs>
  <scannerID>1</scannerID>                         ? Specified Scanner ID
  <cmdArgs>
    <arg-xml>
      <attribute>
        <id>6002</id>                              ? Attribute Number
        <offset>200</offset>                       ? Starting Offset
      </attribute>
    </arg-xml>
  </cmdArgs>
</inArgs>
```

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>
    <response>
      <opcode>5003</opcode>                        ? Method Response Received
      <attrib_list>
        <attribute>
          <id>6002</id>                            ? Attribute Number
          <name></name>
          <datatype>A</datatype>                   ? Attribute Data Type
          <permission>R</permission>               ? Permissions of the Attribute
          <length>3936</length>                    ? Total Length of Value
          <offset>200</offset>                     ? Starting Offset
          <value_length>227</value_length>         ? Length of Received Value
          <value>                                  ? Received Attribute Value
```

(continued on next page)

```
           0x00 0x00 0x00 0x41 0x00 0x00 0x02 0x02 0x00 0x01 0x00 0x0A 0x01 0x0F 0x00 0x00
0x0A 0x14 0x01 0x00 0x01 0x01 0x01 0x0A 0x00 0x0D 0x02 0x01 0x02 0x01 0x08 0xB4 0x4C 0x32
0x03 0x01 0x00 0x00 0x00 0x00 0x02 0x00 0x02 0x0A 0x00 0x01 0x00 0x05 0x05 0x05 0x01 0x0F
0x03 0x21 0x11 0x0E 0x00 0x01 0x1E 0x01 0x0A 0xFF 0x03 0x06 0x00 0x02 0x00 0x03 0x00 0x00
0x00 0x00 0x00 0x01 0x03 0x64 0x01 0x50 0x00 0x32 0x32 0x00 0x08 0x00 0x01 0x00 0x03 0x01
0x01 0x28 0x00 0x00 0x01 0x01 0x00 0x01 0x01 0x02 0x02 0x02 0x02 0x02 0x01 0x02 0x01 0x1E
0x01 0x00 0x01 0x03 0xA0 0x00 0x64 0x00 0x00 0x00 0x00 0x00 0xDF 0x01 0xEF 0x02 0x90 0x01
0x64 0x00 0x4E 0x01 0xFF 0xFF 0xFF 0xFF 0xB4 0x00 0x00 0x00 0xC8 0x00 0x00 0x00 0x06 0x00
0x01 0x00 0x42 0x54 0x53 0x00 0x0C 0x41 0xE5 0xFD 0xF0 0xB6 0xCA 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x07 0x07 0x07 0x07
          </value>
        </attribute>
      </attrib_list>
    </response>
  </arg-xml>
</outArgs>
```

NOTES

- Execution of the method results in sending one `ATTRIBUTE_GET_OFFSET` RMD protocol command via `SSI_MGMT_COMMAND` SSI protocol command.
- The method is supposed to be executed successfully if a required attribute is requested from a scanner through `ATTRIBUTE_GET_OFFSET` RMD command and a response for an issued `ATTRIBUTE_GET_OFFSET` RMD command is received via `SSI_MGMT_COMMAND` SSI packet from a scanner.

ATTR_SET

Set the values of the attribute(s) of a specified scanner. The attribute(s) set using this method is lost after the next power down. Refer to the *Zebra Scanner SDK Attribute Dictionary* (p/n 72E-149786-XX) for attribute numbers, types and possible values.

Opcode

SBT_RSM_ATTR_SET

(continued on next page)

inXML

```
<inArgs>
  <scannerID>1</scannerID>                        ?Specified Scanner ID
  <cmdArgs>
    <arg-xml>
      <attrib_list>
        <attribute>
          <id>1</id>                              ?Attribute Number
          <datatype>F</datatype>                  ?Attribute Data Type
          <value>False</value>                    ?Attribute Value
        </attribute>
        <attribute>
          <id>2</id>                              ?Attribute Number
          <datatype>S</datatype>                  ?Attribute Data Type
          <value>Symbol</value>                   ?Attribute Value
        </attribute>
      </attrib_list>
    </arg-xml>
  </cmdArgs>
</inArgs>
```

outXML

Not used.

✓ ***NOTES***

- Execution of the method results in sending `ATTRIBUTE_SET` RMD protocol command(s) via `SSI_MGMT_COMMAND` SSI protocol command.
- The method is supposed to be executed successfully if `ATTRIBUTE_SET` RMD protocol commands for all required attributes were sent to a scanner and responses for all issued `ATTRIBUTE_SET` RMD commands are received via `SSI_MGMT_COMMAND` SSI packets from a scanner.
- The method supports setting of maximum 100 attributes values. If more than 100 attribute values are requested to be set, the "Execute Command" API returns `SBT_RESULT_INVALID_PARAMS` to indicate that input parameters are not valid.

ATTR_STORE

Store the values of attribute(s) of a specified scanner. Attribute(s) store using this method are persistent over power down and power up cycles. Refer to the *Zebra Scanner SDK Attribute Dictionary* (p/n 72E-149786-XX) for attribute numbers, types and possible values.

Opcode

SBT_RSM_ATTR_STORE

inXML

```
<inArgs>
  <scannerID>1</scannerID>                   ?Specified Scanner ID
  <cmdArgs>
    <arg-xml>
      <attrib_list>
        <attribute>
          <id>1</id>                         ?Attribute Number
          <datatype>F</datatype>             ?Attribute Data Type
          <value>False</value>               ?Attribute Value
        </attribute>
        <attribute>
          <id>2</id>                         ?Attribute Number
          <datatype>S</datatype>             ?Attribute Data Type
          <value>Symbol</value>              ?Attribute Value
        </attribute>
      </attrib_list>
    </arg-xml>
  </cmdArgs>
</inArgs>
```

outXML

Not used.

NOTES

- Execution of the method results in sending ATTRIBUTE_STORE RMD protocol command(s) via SSI_MGMT_COMMAND SSI protocol command.
- The method should execute successfully if ATTRIBUTE_STORE RMD protocol commands for all required attributes were sent to a scanner and responses for all issued ATTRIBUTE_STORE RMD commands are received via SSI_MGMT_COMMAND SSI packets from a scanner.
- The method supports storing of maximum 100 attributes values. If more than 100 attribute values are requested to be stored, the "Execute Command" API returns SBT_RESULT_INVALID_PARAMS to indicate that input parameters are not valid.

INDEX

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

MN001834A04 Revision A - July 2019