RFD8500/i RFID DEVELOPER GUIDE

RFD8500/i RFID DEVELOPER GUIDE

MN002222A03
Revision A
September 2016

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:

Change	Date	Description
-01 Rev A	10/2015	Initial Release
-02 Rev A	3/2016	Software Maintenance Updates
-03 Rev A	9/2016	Updates for the standard RFD8500 (for Rev C software) and RFD8500i; changed references of ’DUT’ to RFD8500.

TABLE OF CONTENTS

ABOUT THIS GUIDE

Introduction

The *RFD8500 RFID Developer Guide* provides installation and programming information for the Software Developer Kit (SDK) that allows RFID application development for Android and iOS devices.

Chapter Descriptions

This guide includes the following topics:

- *Chapter 1, RFD8500 DEVICE OVERVIEW* provides an overview of the RDF8500 device including system requirements, device setup, enabling Bluetooth, pairing information, using the Zebra RFID Mobile application for Android, using the Zebra RFID application for iOS, and using a PC Based Terminal Over ZETI with the RFD8500.
- *Chapter 2, GETTING STARTED with the ZEBRA RFID SDK for iOS* provides instructions for setting up an XCode project to work with the Zebra RFID SDK for iOS.
- *Chapter 3, ZEBRA RFID SDK for iOS* provides detailed information about how to develop iOS applications using the Zebra RFID SDK for iOS.
- *Chapter 4, GETTING STARTED WITH THE ZEBRA RFID DEMO APPLICATION and RFID API3 SDK for ANDROID* provides instruction for importing Zebra RFDI demo application and setting up new Android application project using the Zebra RFID SDK for Android.
- *Chapter 5, ZEBRA RFID SDK for Android* provides detailed information about how to use various functionality of SDK from basic to advance to develop Android application using the Zebra RFID SDK for Android.
- *Chapter 6, ZETI PROGRAMMING GUIDE* provides information for developing applications using the ZETI interface directly.
- *Appendix A, ZETI REFERENCE* provides a ZETI Interface Command Reference table.
- *Appendix B, COMMANDS and ATTRIBUTE REFERENCES* includes commands and attributes.

Related Documents

- Zebra Scanner SDK for Android Developer Guide, p/n MN002223Axx.
- Zebra Scanner SDK for iOS Developer Guide, p/n MN001834Axx.
- Java Class Reference Guide - This guide is in HTML format located under the javadoc directory in the RFID SDK for Android distribution package.
- Zebra RFID SDK for iOS API document (Zebra_Bluetooth_RFID_iOS_SDK_API.pdf) - This document is packaged with Zebra_RFID_SDK_1.0.*.pkg and is located under \Zebra Technologies\RFID SDK\doc\ at the package installation path.
- RFD8500 User Guide, p/n MN002065Axx.
- RFD8500i User Guide, p/n MN-002761-XX.
- RFD8500 Quick Start Guide, p/n MN002225Axx.
- RFD8500i Quick Start Guide, p/n MN-002760-XX
- RFD8500 Regulatory Guide, p/n MN002062Axx.
- RFD8500i Regulatory Guide, p/n MN-002856-xx.
- CRDUNIV-RFD8500-1R Three Slot Universal Charge Only Cradle Regulatory Guide, p/n MN002224Axx.
- RFD8500 Bluetooth Pairing Using S/N Barcode White Paper, available at: www.zebra.com/support.
- Zebra RFD8500 Attribute Data Dictionary, available at: www.zebra.com/support.
- Zebra Scanner SDK Attribute Data Dictionary. p/n 72E-149786-XX.

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 SRFID is used to reference Zebra RFID SDK for iOS APIs via Bluetooth.
- The abbreviation for Bluetooth is BT.
- The acronym *ZETI* is an acronym for Zebra Easy Text Interface.
- *Italics* are used to highlight chapters, sections, screen names, and field names 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 will not result in physical harm to the reader, equipment or data.

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

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

Service Information

If you have a problem using the equipment, contact your facility's technical or systems support. If there is a problem with the equipment, they will contact the Zebra Technologies Global Customer Support Center at: 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 RFD8500 DEVICE OVERVIEW

Introduction

This chapter provides an overview of the RDF8500 device including system requirements, device setup, enabling Bluetooth, pairing information, using the Zebra RFID Mobile application for Android, using the Zebra RFID application for iOS, and using a PC Based Terminal Over ZETI with the RFD8500.

System Requirements

- Developer Computers: Windows 7/64-bit, MacBook Pro with at least 8 Gb of memory.
- Android: Android Studio (1.0 or later), and Android API Level 19 or later. The recommended Android device version is KitKat 4.4.x.
- iOS: iOS SDK 7.0 or later; XCode version 6.0 or later. The recommended iOS version is 8.0 or later. Recommended devices: iPod Touch (5th generation), and iPhone 6.

Setting up the Device

To setup the device:

1. Fully charge the RFD8500 battery by using a USB cable connected to a PC or charger. It is recommended to use a USB power adapter rated at 1.2A**.**

   When the RFD8500 is fully charged the power LED stops blinking and the unit goes into to *Off Mode*.

   IMPORTANT The RFD8500 can not fully boot up if the battery level is low.

2. Disconnect the USB cable and reset the unit by pressing the **Power** button (if the unit is on, press **Power** for 3 seconds to turn it off, and press again to turn it on).
3. Enable Bluetooth®. See *Enabling Bluetooth® (BT)*.
4. To avoid the device moving into low power mode, reconnect the USB cable.

Resetting the Device

- To reset the RFD8500, press the **Power** button for 3 seconds.
- To reset to factory default procedures, press the **Power** and Bluetooth buttons simultaneously for 3 seconds.

Enabling Bluetooth® (BT)

Using Bluetooth on the RFD8500

- The RFD8500 supports a dual SPP port - SSI and RFID serial ports with Android devices.
- The custom UUIDs listed below are exposed for SSI and RFID to be used from the Android device.
  - SSI Custom UUID - 39f8fbf4-dc42-86c6-d163-79a0dcc58858
  - RFID Custom UUID - 2ad8a392-0e49-e52c-a6d2-60834c012263
  - Standard SPP UUID - 00001101-0000-1000-8000-00805F9B34FB.
- BT profiles: SPP, HID, and MFi modes.
- The RFD8500 supports MFi mode (iAP framework) to connect to iOS devices.
- RFID functions are supported using the new ZETI protocol (see *Appendix A, ZETI REFERENCE*). For iOS devices, SDKs, and the Zebra RFID application for iOS are provided using ZETI in MFI mode. For Android devices only the Zebra RFID Mobile application for Android is provided using ZETI in BT SPP mode. The SDK for Android and Zebra RFID Mobile with API showcasing the use of the SDK is also supported.
- Bar code functions are supported using the Simple Serial Interface (SSI) protocol for scanners. For iOS devices, SDKs and the Zebra Scanner Control for iOS application are provided, and it works in MFI mode. For Android devices, an SDK and two Zebra RFID Mobile applications are provided. One of the demo applications uses the ZETI in BT SPP mode, and the other uses the Android APIs. See *Related Documents on page viii* for information on the scanner developer guides.

Pairing with Bluetooth

Prior to pairing, note the following to identify the device:

- Device S/N is printed on the device sticker on the back of the antenna as well as below the battery.

NOTE The RFD8500 requires a physical trigger press for BT pairing to complete. The paring request is visible when a blue BT LED blinks on the RFD8500.

Pairing with Android Devices

1. Go to *Settings* > *Bluetooth* > *Search for devices*.
2. If the BT LED is not blinking, press the BT button for 1 second to make the RFD8500 discoverable (the BT LED starts blinking when in discoverable mode). When the device appears in the list tap the device name.

3. When the BT LED starts to blink rapidly, press the RFD8500 trigger within 25 seconds to accept the pairing request.

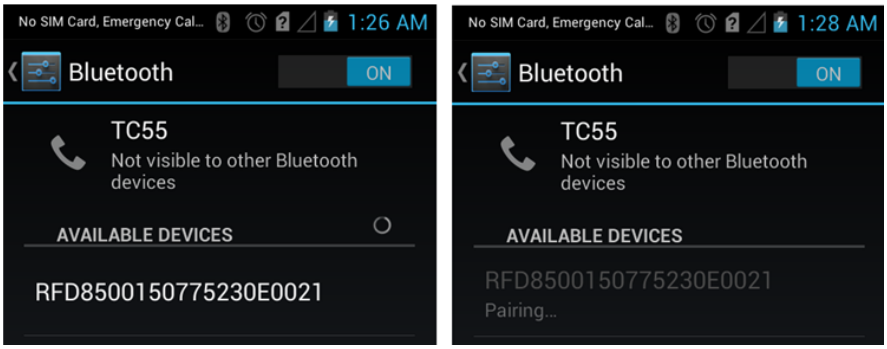

Figure 1-1 *Pairing with an Android Device*

Pairing with a Personal Computer

1. If the BT LED is not blinking, press the BT button for 1 second to make the RFD8500 discoverable (the BT LED starts blinking when in discoverable mode). From the *Start* menu, select *Device and Printers.* Select **Add a device**.

2. Select the device and click **Next**. When the BT LED starts blinking rapidly press the trigger within 25 seconds to acknowledge pairing.

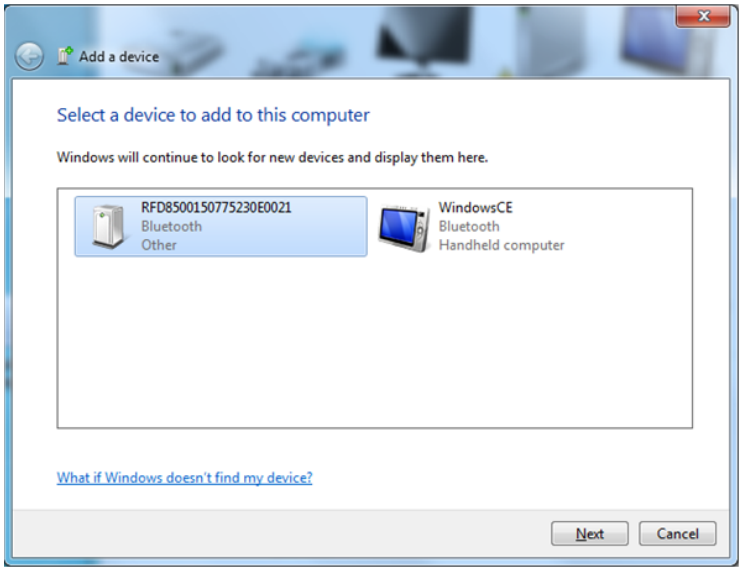

Figure 1-2 *Adding a Device to Pair*

3. Click **Close** to complete the pairing process.

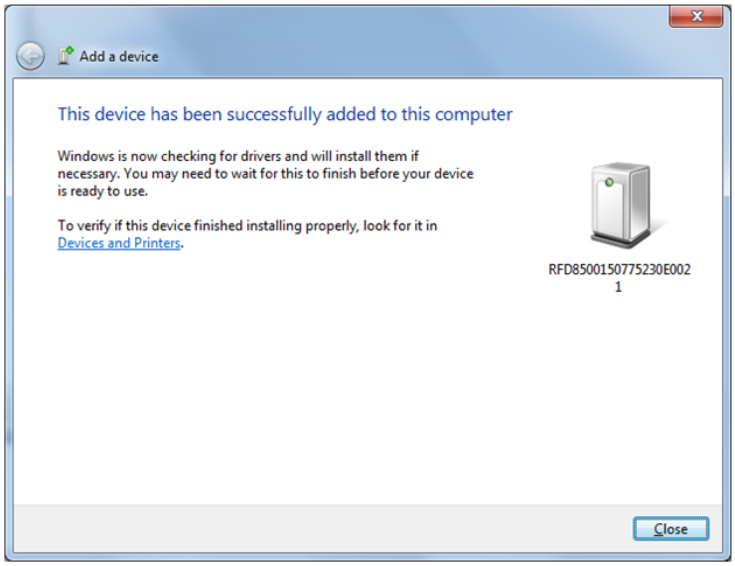

Figure 1-3 *Adding a Device to Pair*

4. When the device is successfully paired, right click to check its properties. Select the *Services* tab and record the assigned COM port number for SPP.

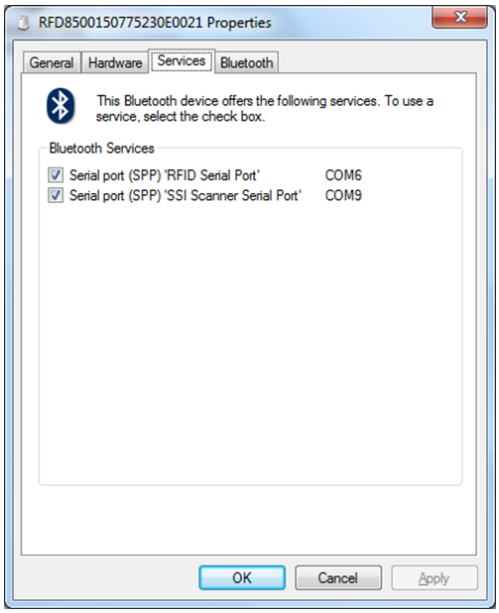

Figure 1-4 *Checking Device Properties*

Using the Zebra RFID Mobile Application for Android with the RFD8500

To use the application with the RFD8500:

1. From *Home* screen go to the *Settings* screen, or using the *Navigation Drawer* go to the *Readers List*.
2. The *Readers List* displays the paired device. Select the RFD8500 reader.

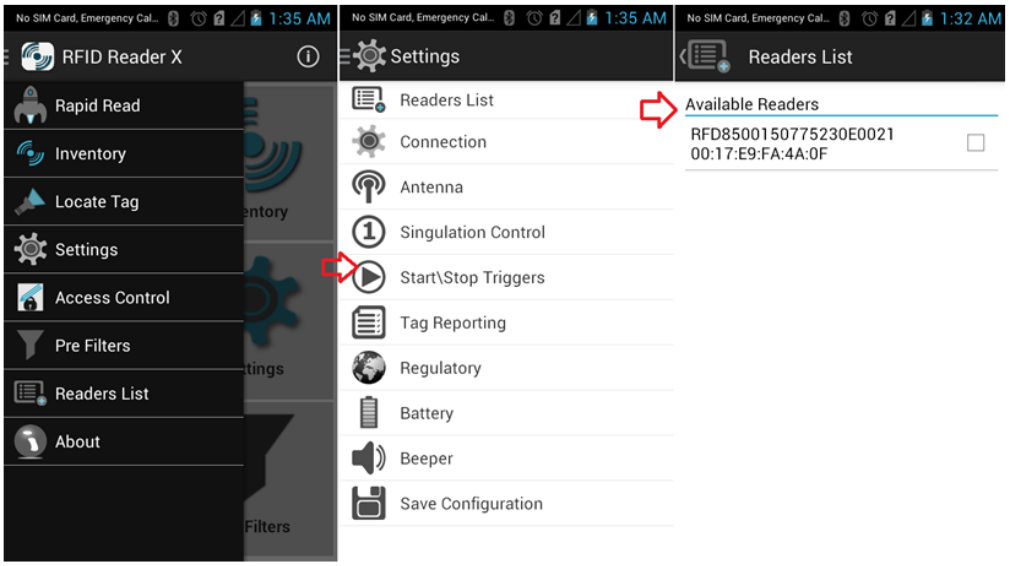

Figure 1-5 *Readers List*

3. When successfully connected, the application displays the regulatory settings. Set the region and the appropriate regulatory information for the RFD8500. Press the back arrow to save the settings.

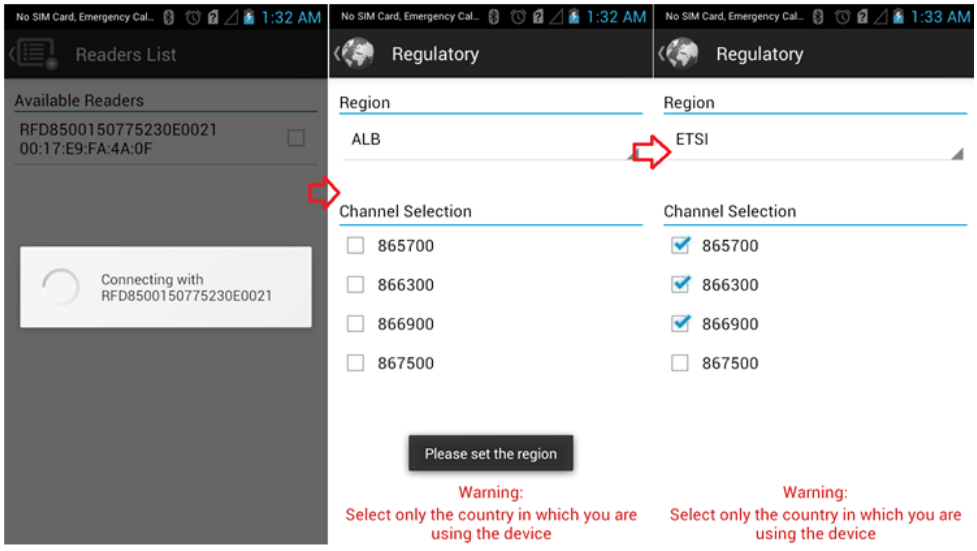

Figure 1-6 *Regulatory Information*

4. The RFD8500 is now ready for RFID operations.
5. Press the back arrow to return to the *Home* screen.

6. Use the *Rapid Read* or *Inventory* screen to read the tags and settings to alter any configurations.

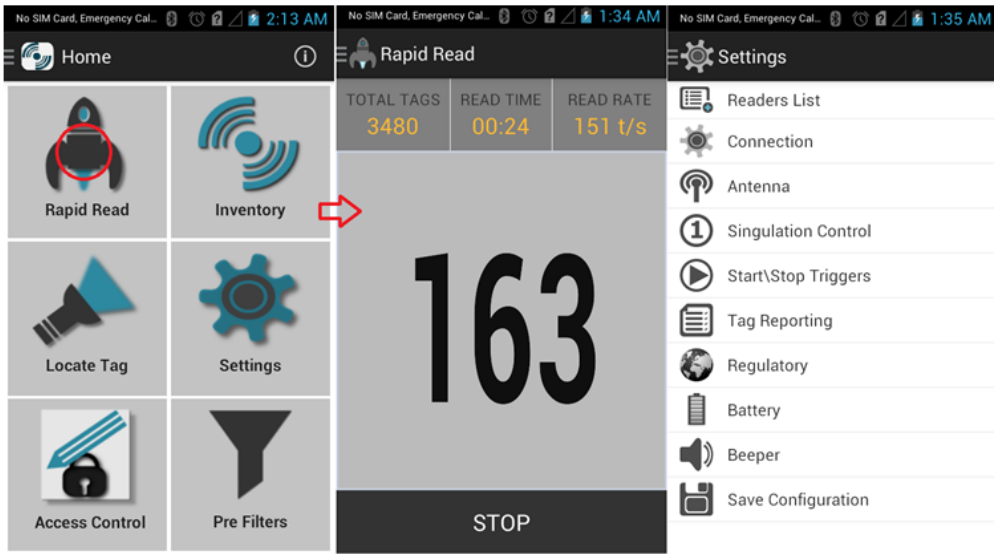

Figure 1-7 *Rapid Read Screen*

NOTE Disconnect the device using the *Reader List* before connecting with another device/PC.

Using the Zebra RFID Application for iOS with iOS Devices

The iOS application must be compiled from sources to be deployed to your iOS devices.

Recommendations:

- iOS version is 8.0 or later.
- Devices - iPod Touch (5th generation); iPhone 6.

Use XCode to open the RFIDDemoApp project (RFIDDemoApp.xcodeproj). From XCode select iPhone6 as the target device and build the project.

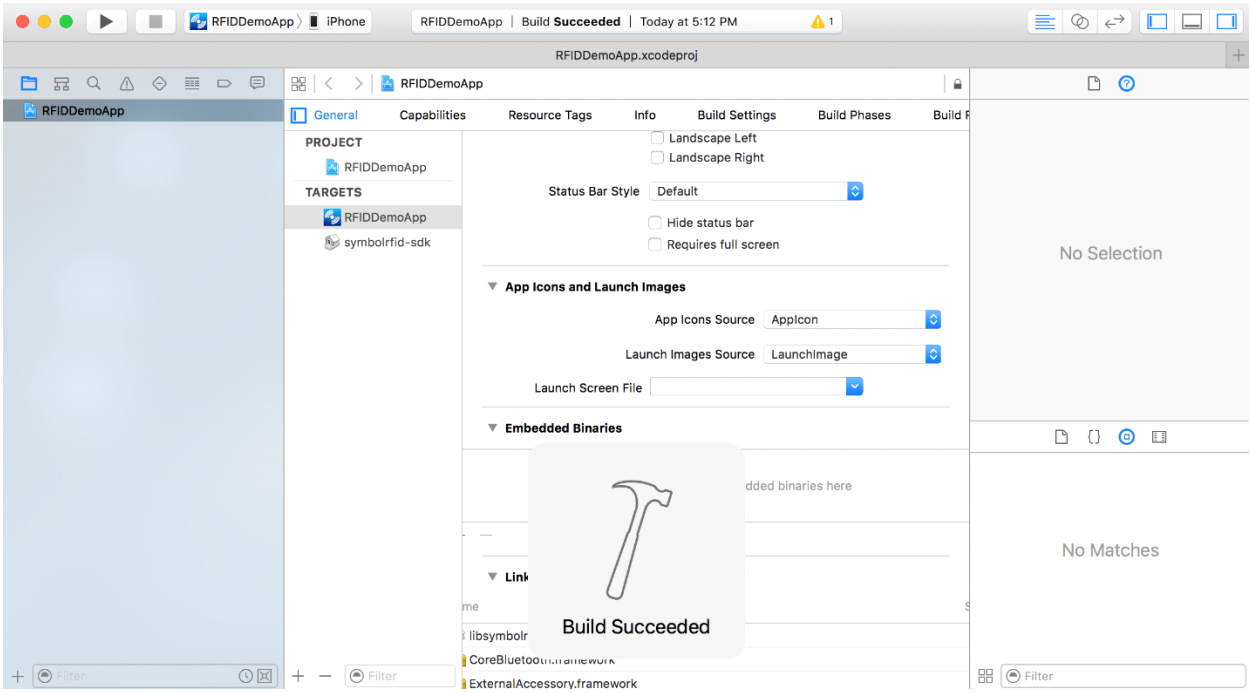

Figure 1-8 *Successful Build Screen*

Pairing with an iOS Device

See *Pairing with Android Devices on page 1-2* for the pairing process. The iOS process is similar to the Android pairing process.

After the application deploys, tap the **About** button to display the *About* screen. The *About* screen displays version information.

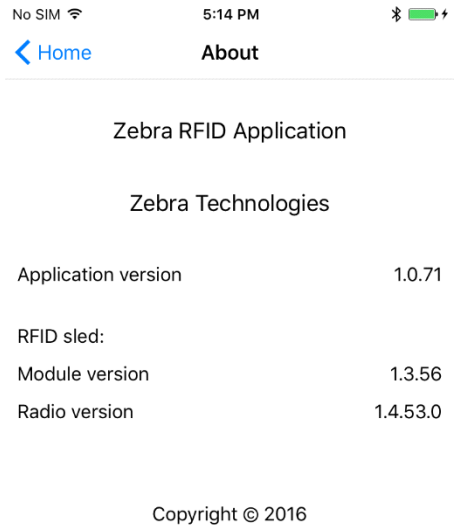

Figure 1-9 *Version Screen*

Pairing using S/N Bar Code

The RFD8500 can be paired with a host that can scan the bar code printed at the bottom of the RFD8500 antenna.

An Android demo application, sample code, and a white paper (see *Related Documents on page viii*) describing this Bluetooth pairing process can be obtained by sending a request to Zebra support.

The procedure generally entails the following, in this order:

1. Run the Android demo application.
2. Press the RFD8500 **Bluetooth** button to start Bluetooth discovery mode.
3. Scan the RFD8500 S/N bar code at the bottom of the RFD8500 antenna.
4. Press the RFD8500 trigger to complete the pairing process.

Using a PC Based Terminal Over ZETI with the RFD8500

1. Open the PC based terminal application.
2. Connect to the COM port identified in *Checking Device Properties on page 1-4*.
3. Run the `cn` command to connect with the RFD8500.
4. Run the `gv` command to get version related information.

   NOTE The region must be set before proceeding to the RFID operation region.

5. Run the `in` command to read the tags.
6. Run the `a` command to abort the operation.

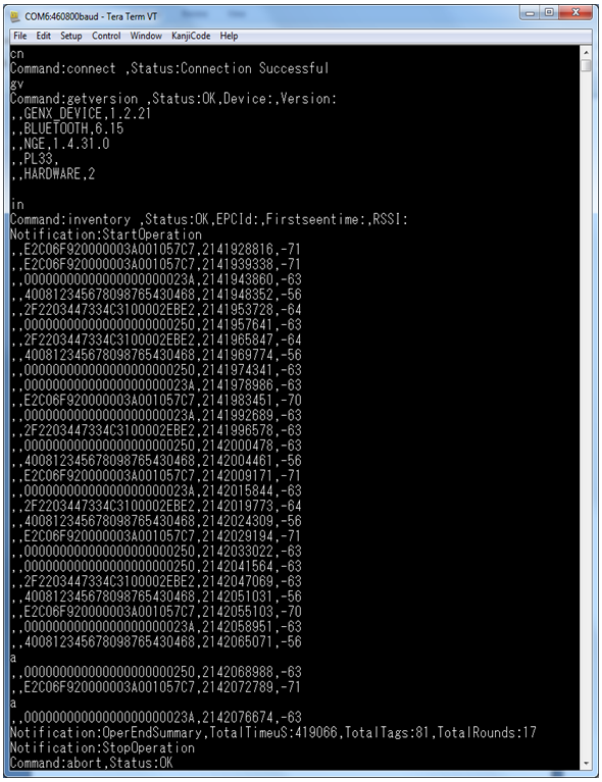

Figure 1-10 *Commands*

Chapter 2 GETTING STARTED with the ZEBRA RFID SDK for iOS

Introduction

This chapter defines step-by-step instructions for setting up a new XCode project to work with the Zebra RFID SDK for iOS.

Setting up an XCode Project for SDK-based iOS Applications

To set up a new XCode project to work with the Zebra RFID SDK for iOS:

1. In XCode IDE, click *Single View Application* to create a new iOS Application project.

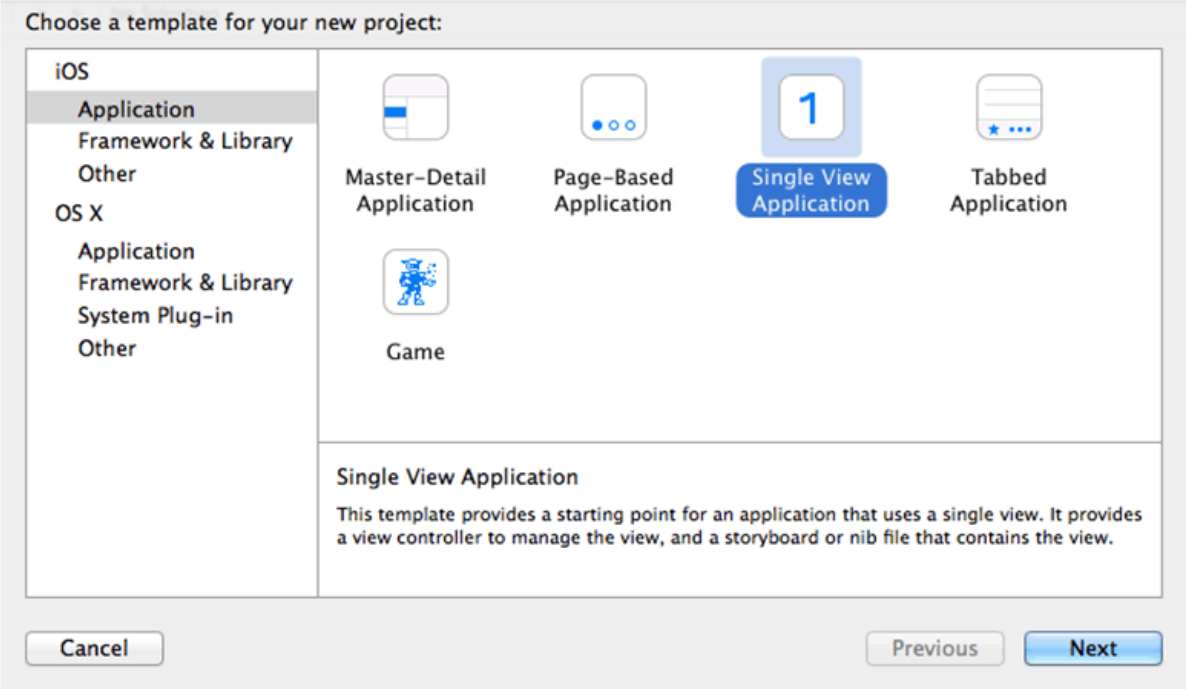

Figure 2-1 *Choosing Project Template*

2. Click **Next**.
3. Choose the options for the project.

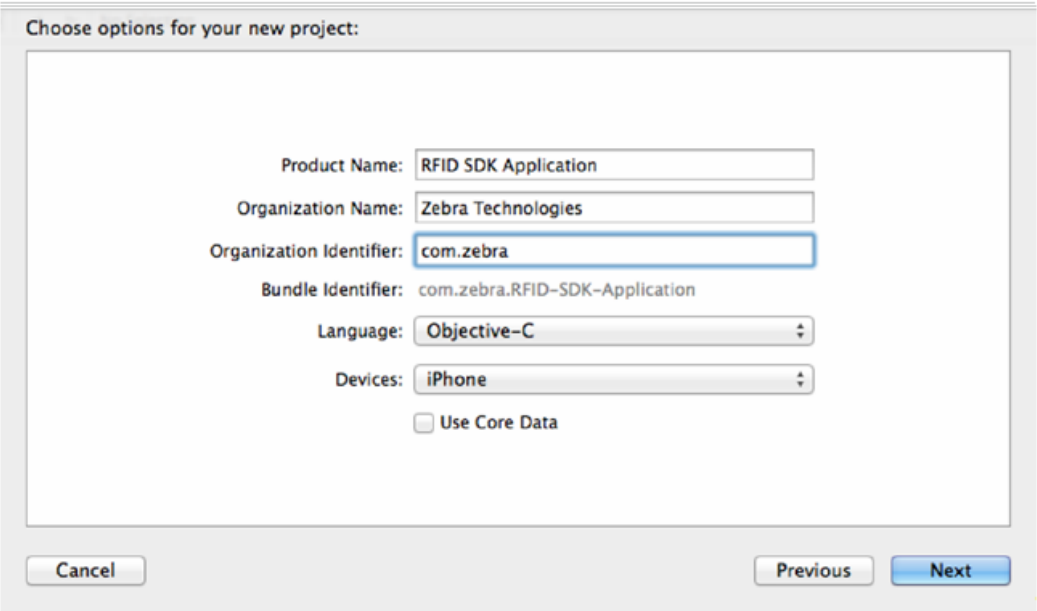

Figure 2-2 *Choosing Project Options*

4. Copy the symbolrfid-sdk folder with static library and headers from the Zebra RFID SDK for iOS installation directory to the root folder of your XCode project.

NOTE Symbolic link can also be used instead of copying.

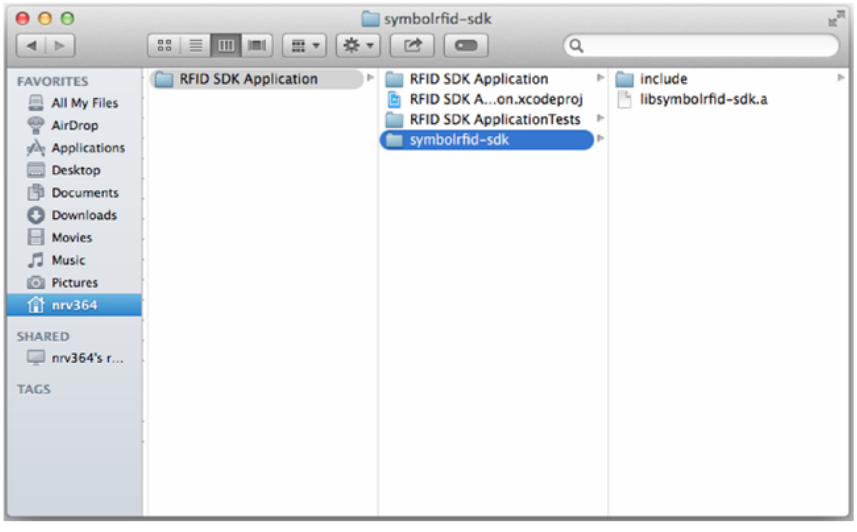

Figure 2-3 *Copying Folder to Project*

5. Configure your XCode project to support the com.zebra.rfd8x00_easytext external accessory communication protocol by including the *UISupportedExternalAccessoryProtocols* key in your application's *Info.plist* file, or via the *[Info]* tab of your project settings.

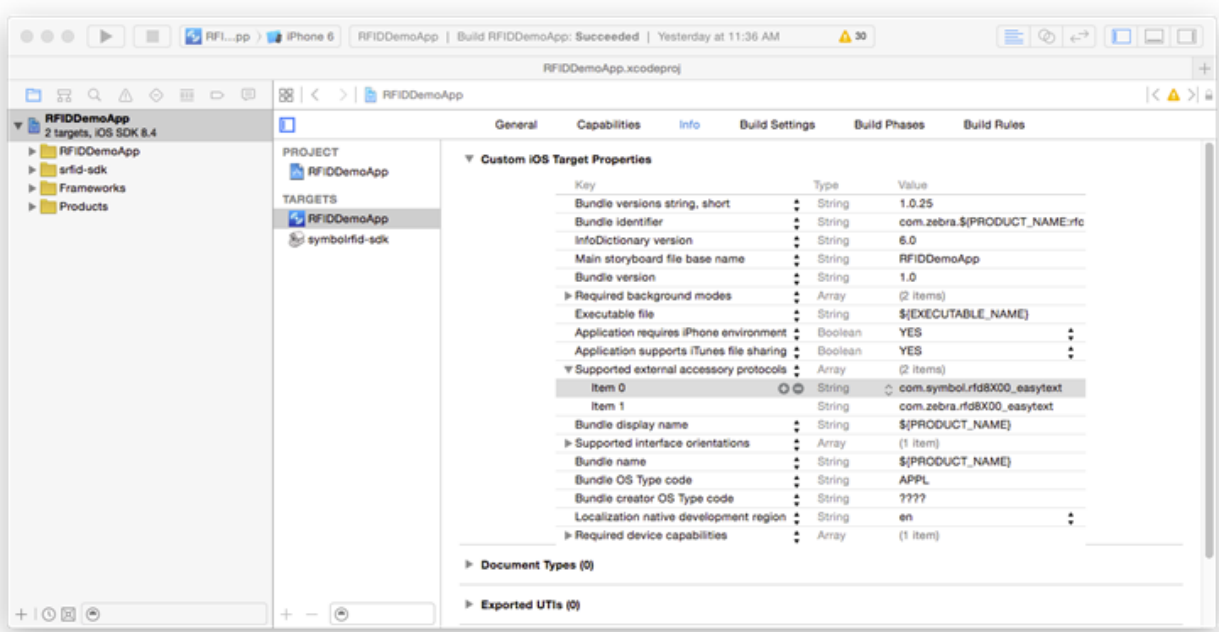

Figure 2-4 *Configure the Supported External Accessory Protocols*

6. If your application is able to communicate with BT RFID readers in a background mode, configure your XCode project to declare the background modes your application supports by including the *UIBackgroundModes* key in your application's *Info.plist* file, or via the *[Info]* tab of your project settings.

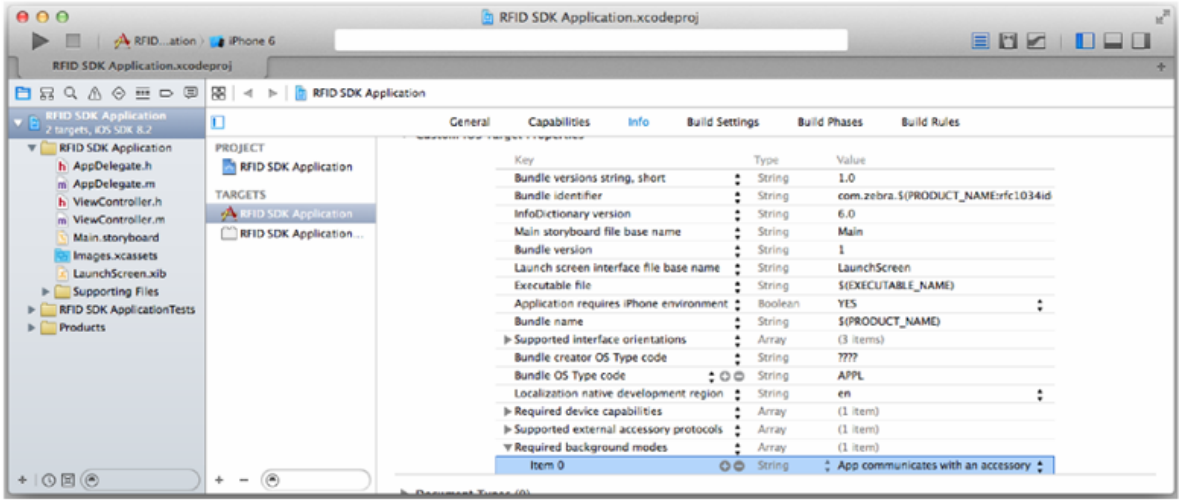

Figure 2-5 *Configure the Required Background Modes*

7. Configure your application to link with the default iOS frameworks listed below that are required for utilization of the Zebra RFID SDK for iOS via *[Link Binary With Libraries]* section of the *[Build Phases]* tab of your project settings.

   - ExternalAccessory.framework

- CoreBluetooth.framework

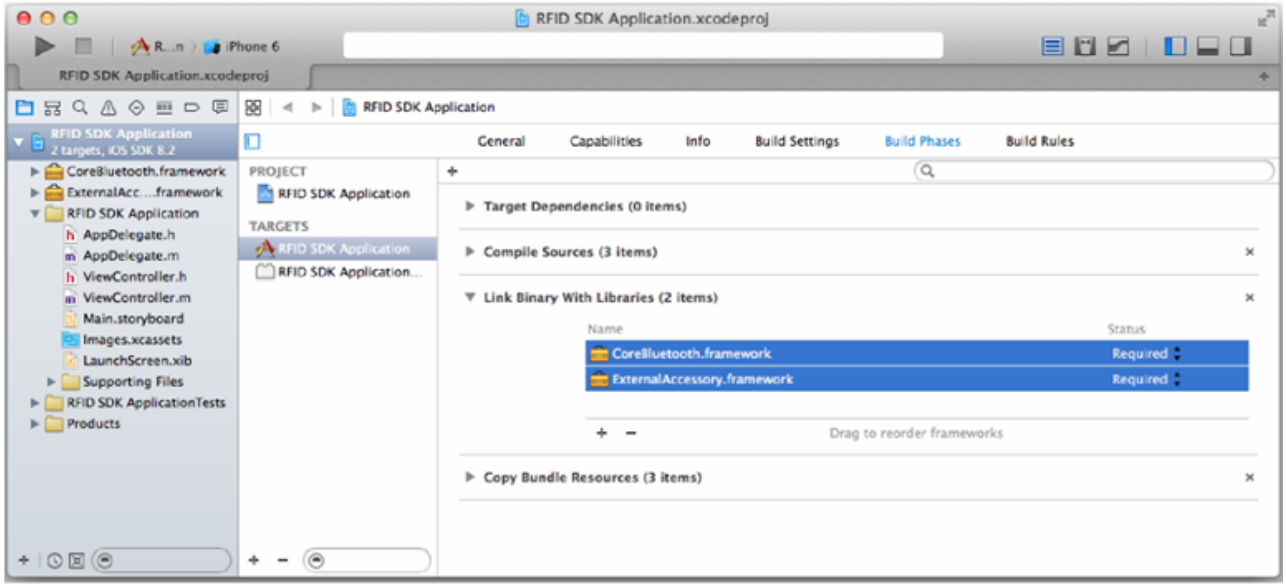

Figure 2-6 *Configure the Linked Libraries*

8. Configure your XCode project to make Zebra RFID SDK for iOS headers available through the *$(SRCROOT)/symbolrfid-sdk/include/* value of the *[User Header Search Paths]* option in the *[Search Paths]* section of the *[Build Settings]* tab of your project settings.

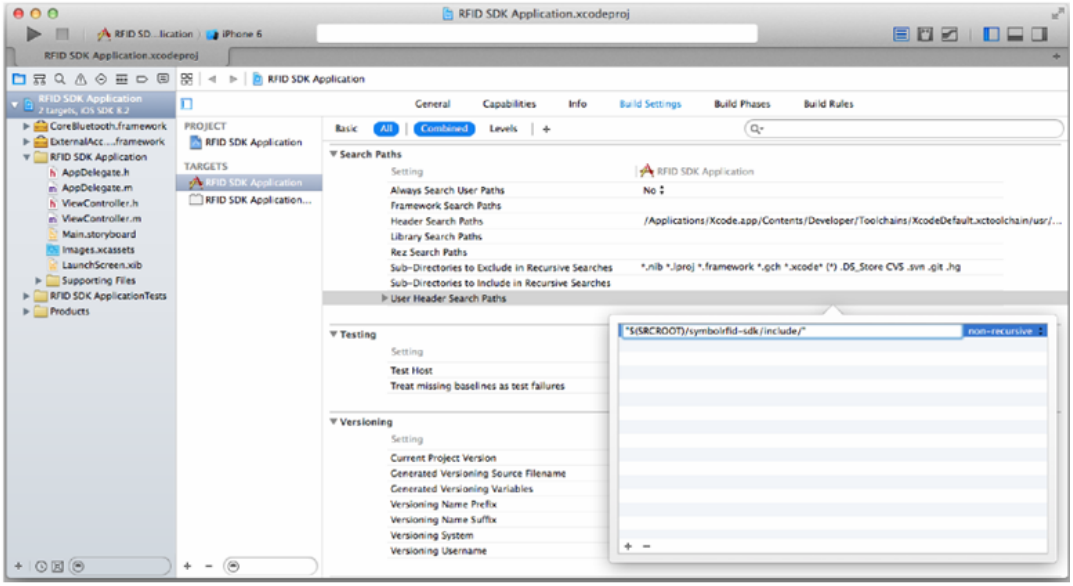

Figure 2-7 *Configure the User Header Search Paths*

9. Configure your application to link with the Zebra RFID SDK for iOS static library through the *[Link Binary With Libraries]* section of the *[Build Phases]* tab of your project settings.

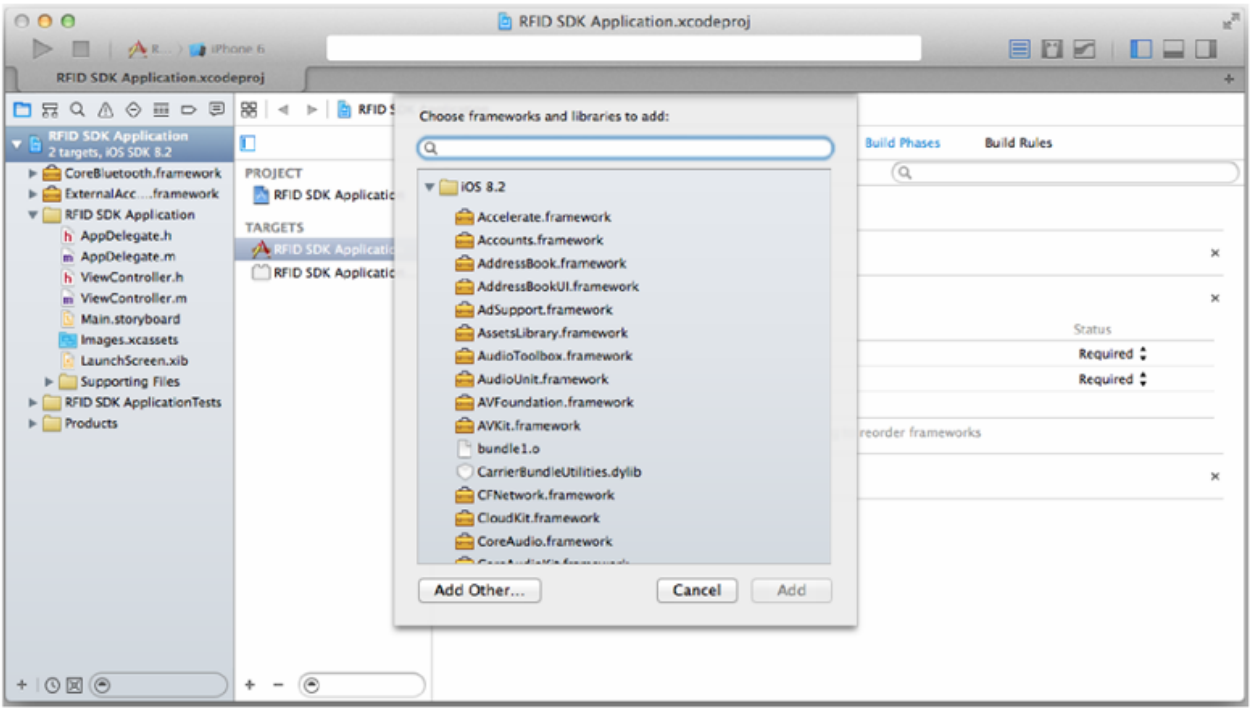

Figure 2-8 *Link Application to the Zebra RFID SDK for iOS Static Library*

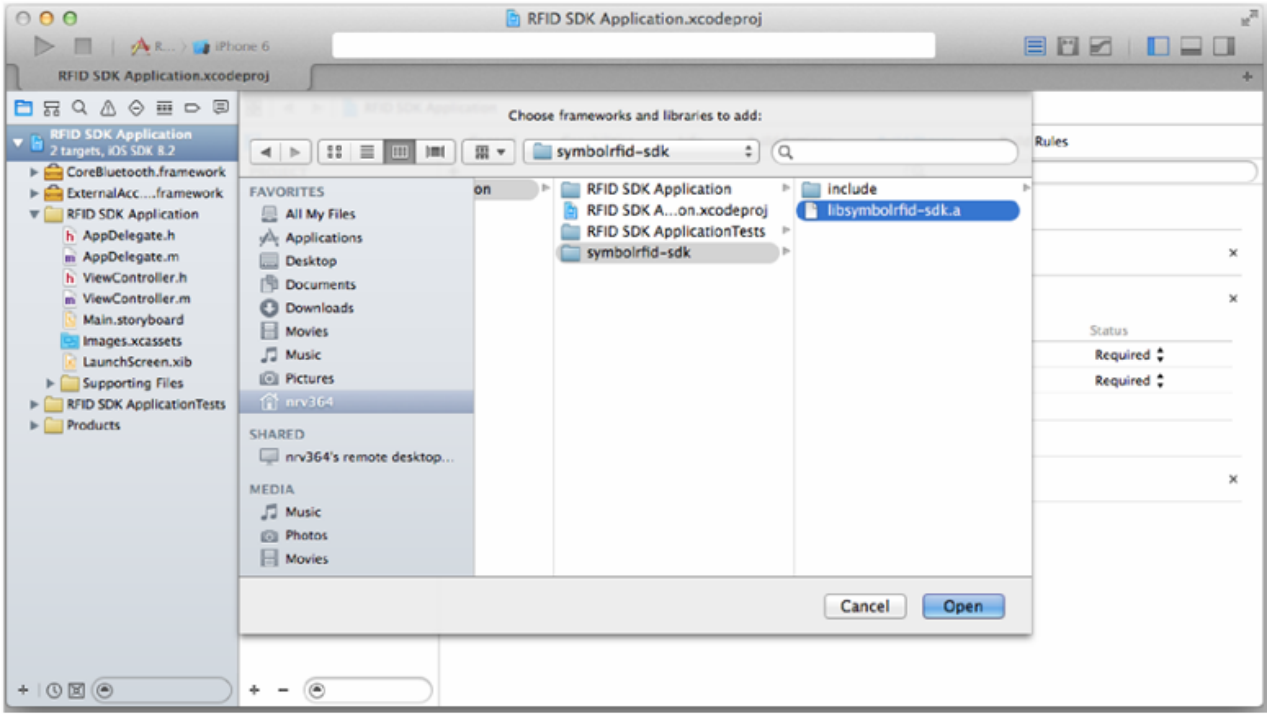

Figure 2-9 *Select the Static Library - libsymbolrfid-sdk.a*

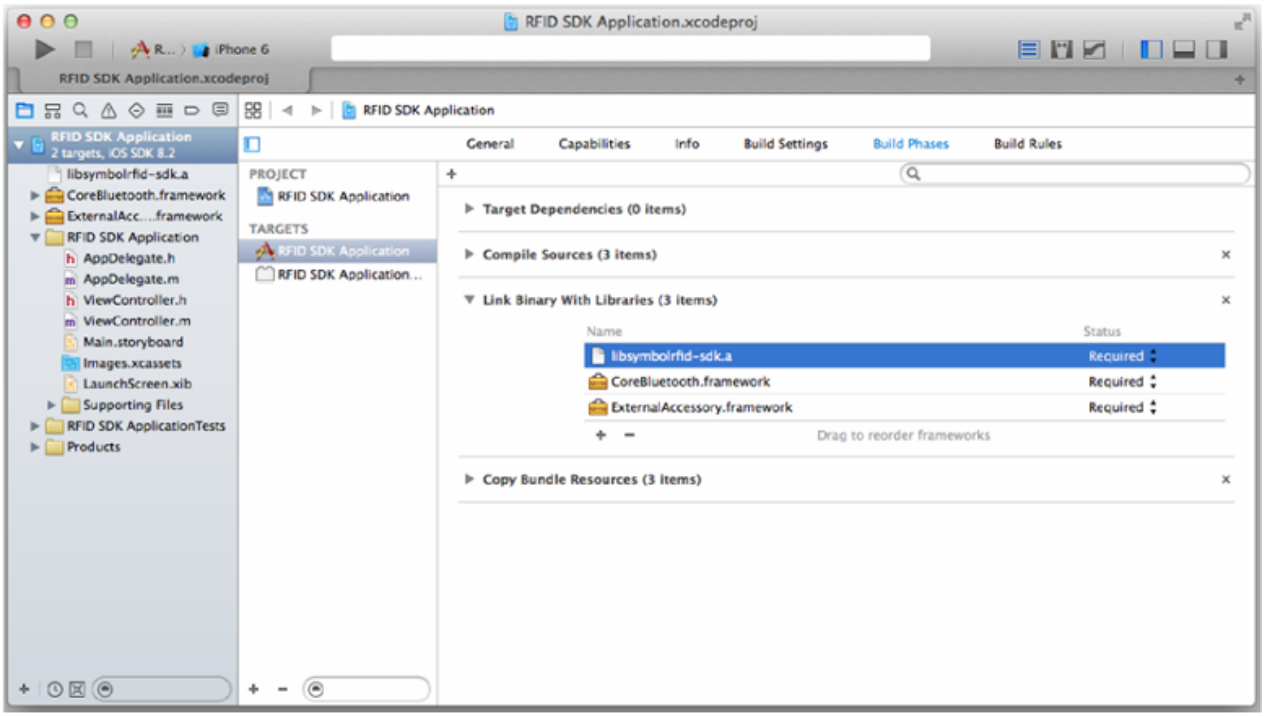

Figure 2-10 *Selected Library in the List of Linked Libraries*

Chapter 3 ZEBRA RFID SDK for iOS

Introduction

This chapter provides detailed information about how to develop iOS applications using the Zebra RFID SDK for iOS.

The Zebra RFID SDK for iOS allows an application to communicate with RFID readers that support the ASCII protocol interface and are connected to an iOS device wirelessly via Bluetooth.

The Zebra RFID SDK for iOS provides the API that can be used by external applications to manage connections of remote RFID readers, and to control connected RFID readers.

RFID SDK Basics

NOTE Detailed API documentation can be found in Zebra_Bluetooth_RFID_iOS_SDK_API.pdf distributed with the Zebra RFID SDK for iOS (see *Related Documents on page viii*.)

The Zebra RFID SDK for iOS is intended for interaction with RFID readers connected to an iOS device via Bluetooth wireless interface. The SDK provided an ability to manage RFID readers' connections, performing various operations with connected RFID readers, configuring connected RFID readers and knowing other information related to connected RFID readers.

The Zebra RFID SDK for iOS consists of a static library that is supposed to be linked with an external iOS application and a set of necessary header files. Step -by-step instructions for configuring XCode project to enable utilization of Zebra RFID SDK for iOS are provided in Getting Started document.

All available API functions are defined by *srfidISdkApi* Objective C protocol. A single shared instance of an API object that implements *srfidISdkApi* protocol can be obtained via *createRfidSdkApiInstance* method of *srfidSdkFactory* class.

```
/* variable to store single shared instance of API object */
id <srfidISdkApi> apiInstance;
/* receiving single shared instance of API object */
apiInstance = [srfidSdkFactory createRfidSdkApiInstance];
/* getting SDK version string */
NSString *sdk_version = [apiInstance srfidGetSdkVersion];
NSLog(@"Zebra SDK version: %@\n", sdk_version);
```

Receiving Asynchronous Notifications from the SDK

The SDK supports a set of asynchronous notifications to inform the application about RFID reader related events (e.g., reception of tag data, starting of radio operation, etc.) and *connectivity* related events (e.g., appearance of RFID reader). All supported callbacks are defined by *srfidISdkApiDelegate* Objective C protocol.

In order to receive asynchronous notification s from the SDK the application performs the following steps.

1. Create an object that implements srfidISdkApiDelegateProtocol.

```
/* definition of class that implements srfidISdkApiDelegate protocol */
@interface EventReceiver : NSObject <srfidISdkApiDelegate> {
     /* variables */
/* methods definition */
```

2. Register the created object as notification receiver via *srfidSetDelegate* API function.

```
/* registration of callback interface with SDK */
EventReceiver *eventListener = [[EventReceiver alloc] init];
apiInstance srfidSetDelegate:eventListener];
```

3. Subscribe for asynchronous event of specific types via srfidSubscribeForEvents API function.

```
/* subscribe for tag data and operation status related events */
[apiInstance srfidSubsribeForEvents:(SRFID_EVENT_MASK_READ |
SRFID_EVENT_MASK_STATUS)];
/* subscribe for battery and hand-held trigger related events */
[apiInstance srfidSubsribeForEvents:(SRFID_EVENT_MASK_BATTERY |
SRFID_EVENT_MASK_TRIGGER)];
```

If a specific object is registered as a notification receiver the SDK calls the corresponding method of the registered object when a specific event occurs if the application is subscribed for events of this type. The SDK may deliver asynchronous events on a main thread or on one of SDK helper threads so the object that implements *srfidISdkApiDelegate* protocol is thread-safe.

Connectivity Management

The Zebra RFID SDK for iOS is designed to support interaction with RFID readers operating in either BT MFi or BT LE mode. The SDK is intentionally configured to enable communication with a specific type of RFID readers via srfidSetOperationalMode API function. If operating mode of the SDK is not configured the SDK remains disabled and is not able to communicate with RFID readers in neither BT MFi nor BT LE modes.

Following example demonstrates enabling interaction with RFID readers in BT MFi mode.

```
/* configuring SDK to communicate with RFID readers in BT MFi mode */
[apiInstance srfidSetOperationalMode:SRFID_OPMODE_MFI];
```

The following terms are introduced to distinguish RFID readers that are seen by the SDK via OS API and RFID readers with which the SDK established a logical communication session and is able to interact. An RFID reader is called available if it is already connected to the iOS device via Bluetooth. The RFID reader is seen by the SDK and the SDK can establish a logical communication session to interact with the RFID reader. If a logical communication session is established with an already connected (via Bluetooth) RFID reader, the RFID reader is called active.

The SDK maintains internal lists of active and available RFID readers. The following example demonstrates reception of lists of active and available RFID readers from the SDK.

(continued on next page)

```
/* allocate an array for storage of list of available RFID readers */
NSMutableArray *available_readers = [[NSMutableArray alloc] init];

/* allocate an array for storage of list of active RFID readers */
NSMutableArray *active_readers = [[NSMutableArray alloc] init];

/* retrieve a list of available readers */
[apiInstance srfidGetAvailableReadersList:&available_readers];

/* retrieve a list of active readers */
[apiInstance srfidGetActiveReadersList:&active_readers];

/* merge active and available readers to a single list */
NSMutableArray *readers = [[NSMutableArray alloc] init];
[readers addObjectsFromArray:active_readers];
[readers addObjectsFromArray:available_readers];
[active_readers release];
[available_readers release];

for (srfidReaderInfo *info in readers)
{
     /* print the information about RFID reader represented by srfidReaderInfo object */
     NSLog(@"RFID reader is %@: ID = %d name = %@\n", (([info isActive] == YES) ? @"active"
: @"available"), [info getReaderID], [info getReaderName]);
}

[readers release];
```

The SDK supports automatic detection of appearance and disappearance of available RFID readers. When the *Available readers detection* option is enabled the SDK updates its internal list of available RFID readers and delivers a corresponding asynchronous notification once it detects connection or disconnection of a specific RFID reader to the iOS device via Bluetooth . If the option is disabled the SDK updates its internal list of available RFID readers only when it is requested by an application via *srfidGetAvailableReadersList* API function. Following example demonstrates enabling of automatic detection and processing of corresponding asynchronous notifications.

```
/* subscribe for connectivity related events */
[apiInstance srfidSubsribeForEvents:(SRFID_EVENT_READER_APPEARANCE |
SRFID_EVENT_READER_DISAPPEARANCE)];
/* configuring SDK to detect appearance and disappearance of available RFID readers */
[apiInstance srfidEnableAvailableReadersDetection:YES];

/* EventReceiver class: partial implementation */
@implementation EventReceiver
...
-(void)srfidEventReaderAppeared:(srfidReaderInfo*)availableReader {
     /* print the information about RFID reader represented by srfidReaderInfo object */
     NSLog(@"RFID reader has appeared: ID = %d name = %@\n", [availableReader getReaderID],
[availableReader getReaderName]);
}

-(void)srfidEventReaderDisappeared:(int)readerID {
NSLog(@"RFID reader has disappeared: ID = %d\n", readerID);
}
...
@end
```

To enable interaction with a specific available RFID reader the application shall establish a logical communication session via srfidEstablishCommunicationSession API function. The SDK will deliver a corresponding asynchronous notification once the logical communication session is established if the application has subscribed for events of this type. To perform various operations with a specific active RFID reader the application shall also establish an ASCII protocol level connection via srfidEstablishAsciiConnection API function. Without an established ASCII protocol level connection most of API functions will fail with a SRFID_RESULT_ASCII_CONNECTION_REQUIRED error code. The interaction with a specific active RFID reader can be terminated by the application via srfidTerminateCommunicationSession API function. When the existing logical communication session is terminated either per application request or due to Bluetooth disconnection the SDK will deliver a corresponding asynchronous notification if the application has subscribed for events of this type. The example on the following page demonstrates establishment of a logical communication session with one of available RFID readers, termination of an existing logical communication session with one of active RFID readers and processing of logical communication session related asynchronous events.

```
/* subscribe for connectivity related events */
[apiInstance srfidSubsribeForEvents:(SRFID_EVENT_SESSION_ESTABLISHMENT |
SRFID_EVENT_SESSION_TERMINATION)];

/* allocate an array for storage of list of available RFID readers */
NSMutableArray *available_readers = [[NSMutableArray alloc] init];
/* retrieve a list of available readers */
[apiInstance srfidGetAvailableReadersList:&available_readers];
if (0 < [available_readers count]) {
     /* at least one available RFID reader exists */
     srfidReaderInfo *reader = (srfidReaderInfo*)[available_readers objectAtIndex:0];
     /* establish logical communication session */

     [apiInstance srfidEstablishCommunicationSession:[reader getReaderID]];
}
[available_readers release];
/* allocate an array for storage of list of active RFID readers */
NSMutableArray *active_readers = [[NSMutableArray alloc] init];
/* retrieve a list of active readers */
[apiInstance srfidGetActiveReadersList:&active_readers];
if (0 < [active_readers count]) {
     /* at least one active RFID reader exists */
     srfidReaderInfo *reader = (srfidReaderInfo*)[active_readers objectAtIndex:0];
     /* terminate logical communication session */
     [apiInstance srfidTerminateCommunicationSession:[reader getReaderID]];
}
[active_readers release];

/* EventReceiver class: partial implementation */
@implementation EventReceiver
...
-(void)srfidEventCommunicationSessionEstablished:(srfidReaderInfo*)activeReader {
    /* print the information about RFID reader represented by srfidReaderInfo object */
    NSLog(@"RFID reader has connected: ID = %d name = %@\n", [activeReader getReaderID],
[activeReader getReaderName]);
```

(continued on next page)

```
    /* establish an ASCII protocol level connection */
    NSString *password = @"ascii password";
    SRFID_RESULT result = [apiInstance srfidEstablishAsciiConnection:[reader getReaderID]
aPassword:password];
    if (SRFID_RESULT_SUCCESS == result) {
        NSLog(@"ASCII connection has been established\n");
    }
    else if (SRFID_RESULT_WRONG_ASCII_PASSWORD == result) {
        NSLog(@"Incorrect ASCII connection password\n");
    }
    else {
        NSLog(@"Failed to establish ASCII connection\n");
    }
}
-(void)srfidEventCommunicationSessionTerminated:(int)readerID
     { NSLog(@"RFID reader has disconnected: ID = %d\n", readerID);
}
...
@end
```

The SDK supports **Automatic communication session reestablishment** option. When the option is enabled the SDK automatically establishes a logical communication session with the last active RFID reader that had unexpectedly disappeared once the RFID reader is recognized as available. If the *Available readers detection* option is enabled the RFID reader is recognized as available automatically when it becomes connected via Bluetooth. Otherwise, the SDK adds the RFID reader to the list of available RFID readers only during discovery procedure requested by the application via *srfidGetAvailableReadersList* API. The option has no effect if the application has intentionally terminate a communication session with the active RFID reader via *srfidTerminateCommunicationSession* API function. The *Automatic communication session reestablishment* option is configured via the *srfidEnableAutomaticSessionReestablishment* API function.

```
/* enable automatic communication session reestablishment */
[apiInstance srfidEnableAutomaticSessionReestablishment:YES];
```

Knowing the Reader Related Information

Knowing the Software Version

The SDK provides an ability to retrieve information about software versions of various components of a specific active RFID reader. Software version related information could be retrieved via *srfidGetReaderVersionInfo* API function as demonstrated in the following example.

```
/* identifier of one of active RFID readers is supposed to be stored in m_ReaderId variable */
/* allocate object for storage of version related information */
srfidReaderVersionInfo *version_info = [[srfidReaderVersionInfo alloc] init];

/* an object for storage of error response received from RFID reader */
NSString *error_response = nil;
```

(continued on next page)

```
/* retrieve version related information */
SRFID_RESULT result = [apiInstance srfidGetReaderVersionInfo:m_ReaderId
aReaderVersionInfo:&version_info aStatusMessage:&error_response];

if (SRFID_RESULT_SUCCESS == result) {
     /* print the received version related information */
     NSLog(@"Device version: %@\n", [version_info getDeviceVersion]);
     NSLog(@"NGE version: %@\n", [version_info getNGEVersion]);
     NSLog(@"Bluetooth version: %@\n", [version_info getBluetoothVersion]);
}
else if (SRFID_RESULT_RESPONSE_ERROR == result) {
     NSLog(@"Error response from RFID reader: %@\n", error_response);
}
else if (SRFID_RESULT_RESPONSE_TIMEOUT == result) {
     NSLog(@"Timeout occurs during communication with RFID reader\n");
}
else if (SRFID_RESULT_READER_NOT_AVAILABLE == result) {
     NSLog(@"RFID reader with id = %d is not available\n", m_ReaderId);
}
else {
     NSLog(@"Request failed\n");
}

[version_info release];
```

Knowing the Reader Capabilities

The SDK provides an ability to retrieve the capabilities (or read-only properties) of a specific active RFID reader.

The reader capabilities include the following:

- Serial number
- Model name
- Manufacturer
- Manufacturing date.
- Device name
- ASCII protocol version
- Number of select records (pre-filters)
- Minimal and maximal antenna power levels (in 0.1 dBm units)
- Step for configuration of antenna power level (in 0.1 dBm units)
- Version of air protocol
- Bluetooth address
- Maximal number of operations to be combined in a sequence.

The reader capabilities could be retrieved via *srfidGetReaderCapabilitiesInfo* API function as demonstrated in the following example.

```
/* identifier of one of active RFID readers is supposed to be stored in m_ReaderId variable */

/* allocate object for storage of capabilities information */
srfidReaderCapabilitiesInfo *capabilities = [[srfidReaderCapabilitiesInfo alloc] init];

/* an object for storage of error response received from RFID reader */
NSString *error_response = nil;

/* retrieve capabilities information */
SRFID_RESULT result = [apiInstance srfidGetReaderCapabilitiesInfo:m_ReaderId
aReaderCapabilitiesInfo:&capabilities   aStatusMessage:&error_response];

if (SRFID_RESULT_SUCCESS == result) {

/* print the received capabilities related information */
     NSLog(@"Serial number: %@\n", [capabilities getSerialNumber]);
     NSLog(@"Model: %@\n", [capabilities getModel]);
     NSLog(@"Manufacturer: %@\n", [capabilities getManufacturer]);
     NSLog(@"Manufacturing date: %@\n", [capabilities getManufacturingDate]);
     NSLog(@"Scanner name: %@\n", [capabilities getScannerName]);
     NSLog(@"Ascii version: %@\n", [capabilities getAsciiVersion]);
     NSLog(@"Air version: %@\n", [capabilities getAirProtocolVersion]);
     NSLog(@"Bluetooth address: %@\n", [capabilities getBDAddress]);
     NSLog(@"Select filters number: %d\n", [capabilities getSelectFilterNum]);
     NSLog(@"Max access sequence: %d\n", [capabilities getMaxAccessSequence]);
     NSLog(@"Power level: min = %d; max = %d; step = %d\n", [capabilities

getMinPower], [capabilities getMaxPower], [capabilities getPowerStep]);
}
else if (SRFID_RESULT_RESPONSE_ERROR == result) {
     NSLog(@"Error response from RFID reader: %@\n", error_response);
}
else if (SRFID_RESULT_RESPONSE_TIMEOUT == result) {
     NSLog(@"Timeout occurs during communication with RFID reader\n");
}
else if (SRFID_RESULT_READER_NOT_AVAILABLE == result) {
     NSLog(@"RFID reader with id = %d is not available\n", m_ReaderId);
}
else {
     NSLog(@"Request failed\n");
}

[capabilities release];
```

Knowing Supported Regions

The RFID reader could be configured to operate in a various countries. The SDK provides an ability to retrieve the list of regions supported by a specific active RFID reader.

The list of supported regions could be retrieved via *srfidGetSupportedRegions* API function as demonstrated in the following example.

```
/* identifier of one of active RFID readers is supposed to be stored in m_ReaderId variable */

/* allocate object for storage of region information */
NSMutableArray *regions = [[NSMutableArray alloc] init];
/* an object for storage of error response received from RFID reader */
NSString *error_response = nil;
/* retrieve supported regions */
SRFID_RESULT result = [apiInstance srfidGetSupportedRegions:m_ReaderId
aSupportedRegions:&regions aStatusMessage:&error_response];
if (SRFID_RESULT_SUCCESS == result) {
     /* print supported regions information */
     NSLog(@"Number of supported regions: %d\n", [regions count]);
     for (srfidRegionInfo *info in regions)
     {
         NSLog(@"Regions [%@] is supported: %@\n", [info getRegionName], [info
getRegionCode]);
     }
}
else if (SRFID_RESULT_RESPONSE_ERROR == result) {
     NSLog(@"Error response from RFID reader: %@\n", error_response);
}
else if (SRFID_RESULT_RESPONSE_TIMEOUT == result) {
     NSLog(@"Timeout occurs during communication with RFID reader\n");
}
else if (SRFID_RESULT_READER_NOT_AVAILABLE == result) {
     NSLog(@"RFID reader with id = %d is not available\n", m_ReaderId);
}
else {
     NSLog(@"Request failed\n");
}
[regions release];
```

As the RFID reader could be configured to operate on a specific radio channels in some of countries the SDK provides an ability to retrieve the detailed information regarding one of regions supported by a specific active RFID reader. The detailed information includes a set of channel supported in the region and allowance of hopping configuration.

This information could be retrieved via *srfidGetRegionInfo* API function as demonstrated in the following example.

```
/* identifier of one of active RFID readers is supposed to be stored in m_ReaderId variable */

/* allocate object for storage of supported channels information */
NSMutableArray *channels = [[NSMutableArray alloc] init];
BOOL hopping = NO;
/* an object for storage of error response received from RFID reader */
NSString *error_response = nil;
/* retrieve detailed information about region specified by "USA" region code */
SRFID_RESULT result = [apiInstance srfidGetRegionInfo:m_ReaderId aRegionCode:@"USA"
aSupportedChannels:&channels aHoppingConfigurable:&hopping
aStatusMessage:&error_response];

if (SRFID_RESULT_SUCCESS == result) {
     /* print retrieved detailed information */
     NSLog(@"Hopping configuration is: %@\n", ((YES == hopping) ? @"supported" :
@"NOT supported"));

     for (NSString *str_channel in channels)
     {
         NSLog(@"Supported channel: %@\n", str_channel);
     }
}
else if (SRFID_RESULT_RESPONSE_ERROR == result) {
     NSLog(@"Error response from RFID reader: %@\n", error_response);
}
else if (SRFID_RESULT_RESPONSE_TIMEOUT == result) {
     NSLog(@"Timeout occurs during communication with RFID reader\n");
}
else if (SRFID_RESULT_READER_NOT_AVAILABLE == result) {
     NSLog(@"RFID reader with id = %d is not available\n", m_ReaderId);
}
else {
     NSLog(@"Request failed\n");
}

[channels release];
```

Knowing Supported Link Profiles

An antenna of the RFID reader could be configured to operate in various RF modes (link profiles). The SDK provides an ability to retrieved the list of link profiles (RF modes) supported by a specific active RFID reader.

The list of supported link profiles could be retrieved via *srfidGetSupportedLinkProfiles* API function as demonstrated in the following example.

```
/* identifier of one of active RFID readers is supposed to be stored in m_ReaderId variable */

/* allocate object for storage of link profiles information */
NSMutableArray *profiles = [[NSMutableArray alloc] init];
/* an object for storage of error response received from RFID reader */
NSString *error_response = nil;

/* retrieve supported link profiles */
SRFID_RESULT result = [apiInstance srfidGetSupportedLinkProfiles:m_ReaderId
aLinkProfilesList:&profiles aStatusMessage:&error_response];

if (SRFID_RESULT_SUCCESS == result) {
     /* print retrieved information about supported link profiles */
     NSLog(@"Number of supported link profiles: %d\n", [profiles count]);
     for (srfidLinkProfile *profile_info in profiles) {
         NSLog(@"RF mode index: %d\n", [profile_info getRFModeIndex]);
         NSLog(@"BDR: %d\n", [profile_info getBDR]);
         NSLog(@"PIE: %d\n", [profile_info getPIE]);
         NSLog(@"Tari: min = %d; max = %d; step = %d\n", [profile_info
getMinTari], [profile_info getMaxTari], [profile_info getStepTari]);
         NSLog(@"EPCHAGT&CConformance: %@\n", ((NO == [profile_info
getEPCHAGTCConformance]) ? @"NO" : @"YES"));
         NSLog(@"Divide Ratio: %@\n", [profile_info getDivideRatioString]);
         NSLog(@"FLM: %@\n", [profile_info getForwardLinkModulationString]);
         NSLog(@"M: %@\n", [profile_info getModulationString]);
         NSLog(@"Spectral Mask indicator: %@\n", [profile_info
getSpectralMaskIndicatorString]);
     }
}
else if (SRFID_RESULT_RESPONSE_ERROR == result) {
     NSLog(@"Error response from RFID reader: %@\n", error_response);
}
else if (SRFID_RESULT_RESPONSE_TIMEOUT == result) {
     NSLog(@"Timeout occurs during communication with RFID reader\n");
}
else if (SRFID_RESULT_READER_NOT_AVAILABLE == result) {
     NSLog(@"RFID reader with id = %d is not available\n", m_ReaderId);
}
else {
     NSLog(@"Request failed\n");
}

[profiles release];
```

Knowing Battery Status

A specific active RFID reader could send an asynchronous notification regarding battery status. The SDK informs the application about received asynchronous battery status event if the application has subscribed for events of this type. The SDK also provides an ability to cause a specific active RFID reader to immediately send information about current battery status.

The following example demonstrates both requesting and processing of asynchronous battery status related notifications.

```
/* subscribe for battery related events */
[apiInstance srfidSubsribeForEvents:SRFID_EVENT_MASK_BATTERY];
/* identifier of one of active RFID readers is supposed to be stored in m_ReaderId variable */
/* cause RFID reader to generate asynchronous battery status notification */
SRFID_RESULT result = [apiInstance srfidRequestBatteryStatus:m_ReaderId];
if (SRFID_RESULT_SUCCESS == result)
{
     NSLog(@"Request succeed\n");
}
else {
     NSLog(@"Request failed\n");
}
/* EventReceiver class: partial implementation */
@implementation EventReceiver
...
-    (void)srfidEventBatteryNotity:(int)readerID
aBatteryEvent:(srfidBatteryEvent*)batteryEvent {
     /* print the received information regarding battery status */
     NSLog(@"Battery status event received from RFID reader with ID = %d\n",
readerID);
     NSLog(@"Battery level: %d\n", [batteryEvent getPowerLevel]);
     NSLog(@"Charging: %@\n", ((NO == [batteryEvent getIsCharging]) ? @"NO" :
@"YES"));
     NSLog(@"Event cause: %@\n", [batteryEvent getEventCause]);
}
...
@end
```

Configuring the Reader

The Zebra RFID SDK for iOS API supports managing of various RFID reader parameters including:

- Antenna parameters
- Singulation parameters
- Start and stop triggers parameters
- Tag report parameters
- Regulatory parameters
- Pre-filters
- Beeper.

Antenna Configuration

The following antenna related settings could be configured via the SDK:

- Output power level (in 0.1 dBm units)
- Index of selected link profile (RF mode)
- Application of pre-filters (select records)
- Tari (Type-A reference interval).

Tari value is set in accordance with the selected link profile, (i.e., tari value is in the interval between minimal and maximal tari values specified by the selected link profile). If step size is supported by the selected link profile, the tari value must be a multiple of step size. Antenna settings could be retrieved and set via *srfidGetAntennaConfiguration* and *srfidSetAntennaConfiguration* API function accordingly.

Following example demonstrates retrieving current antenna settings and setting of antenna configuration with minimal output power and one of supported link profiles.

```
/* allocate an array for storage of list of active RFID readers */
NSMutableArray *active_readers = [[NSMutableArray alloc] init];
/* retrieve a list of active readers */
[apiInstance srfidGetActiveReadersList:&active_readers];

if (0 < [active_readers count]) {
     /* at least one active RFID reader exists */
     srfidReaderInfo *reader = (srfidReaderInfo*)[active_readers objectAtIndex:0];
     int reader_id = [reader getReaderID];

     /* allocate object for storage of antenna settings */
     srfidAntennaConfiguration *antenna_cfg = [[srfidAntennaConfiguration alloc]
init];

     /* an object for storage of error response received from RFID reader */
     NSString *error_response = nil;
     /* retrieve antenna configuration */
     SRFID_RESULT result = [apiInstance srfidGetAntennaConfiguration:reader_id
aAntennaConfiguration:&antenna_cfg aStatusMessage:&error_response];
     if (SRFID_RESULT_SUCCESS == result) {
     /* antenna configuration received */
         NSLog(@"Antenna power level: %1.1f\n", [antenna_cfg getPower]/10.0);
         NSLog(@"Antenna RF mode index: %d\n", [antenna_cfg getLinkProfileIdx]);
         NSLog(@"Antenna tari: %d\n", [antenna_cfg getTari]);
         NSLog(@"Antenna pre-filters application: %@", ((NO == [antenna_cfg
getDoSelect]) ? @"NO" : @"YES"));
     }
     else if (SRFID_RESULT_RESPONSE_ERROR == result) {
         NSLog(@"Error response from RFID reader: %@\n", error_response);
     }
     else if (SRFID_RESULT_RESPONSE_TIMEOUT == result) {
         NSLog(@"Timeout occurs during communication with RFID reader\n");
}
     else if (SRFID_RESULT_READER_NOT_AVAILABLE == result) {
         NSLog(@"RFID reader with id = %d is not available\n", m_ReaderId);
     }
     else {
```

(continued on next page)

```
        NSLog(@"Request failed\n");
    }

    [antenna_cfg release]; error_response = nil;

    /* RF mode index to be set */
    int link_profile_idx = 0;
    /* tari to be set */
    int tari = 0;
    /* 20.0 dbm power level to be set */
    int power = 200;

    /* allocate object for storage of link profiles information */
    NSMutableArray *profiles = [[NSMutableArray alloc] init];

    /* retrieve supported link profiles */
    result = [apiInstance srfidGetSupportedLinkProfiles:reader_id
aLinkProfilesList:&profiles aStatusMessage:&error_response];

    if (SRFID_RESULT_SUCCESS == result) {
        if (0 < [profiles count]) {
            srfidLinkProfile *profile = (srfidLinkProfile*)[profiles lastObject];
            link_profile_idx = [profile getRFModeIndex];
            tari = [profile getMaxTari];
        }
    }

    [profiles release];

    /* allocate object for storage of capabilities information */
    srfidReaderCapabilitiesInfo *capabilities = [[srfidReaderCapabilitiesInfo alloc]
init];
    /* retrieve capabilities information */
    result = [apiInstance srfidGetReaderCapabilitiesInfo:reader_id
aReaderCapabilitiesInfo:&capabilities   aStatusMessage:&error_response];
    if (SRFID_RESULT_SUCCESS == result) {
        power = [capabilities getMinPower];
    }

    [capabilities release];

    /* prepare an object with desired antenna parameters */
    antenna_cfg = [[srfidAntennaConfiguration alloc] init];
    [antenna_cfg setLinkProfileIdx:link_profile_idx];
    [antenna_cfg setPower:power];
    [antenna_cfg setTari:tari];
    [antenna_cfg setDoSelect:NO];

    error_response = nil;
    /* set antenna configuration */
    result = [apiInstance srfidSetAntennaConfiguration:reader_id
aAntennaConfiguration:antenna_cfg aStatusMessage:&error_response];

    [antenna_cfg release];
```

(continued on next page)

```
    if (SRFID_RESULT_SUCCESS == result) {
        /* antenna configuration applied successfully */
        NSLog(@"Antenna configuration has been set\n");
    }
    else if (SRFID_RESULT_RESPONSE_ERROR == result) {
        NSLog(@"Error response from RFID reader: %@\n", error_response);
    }
    else if (SRFID_RESULT_RESPONSE_TIMEOUT == result) {
        NSLog(@"Timeout occurs during communication with RFID reader\n");
    }
    else if (SRFID_RESULT_READER_NOT_AVAILABLE == result) {
        NSLog(@"RFID reader with id = %d is not available\n", m_ReaderId);
    }
    else {
        NSLog(@"Request failed\n");
    }
}
else {
    NSLog(@"No active RFID readers\n");
}
[active_readers release];
```

Singulation Configuration

Following singulation control settings could be configured via the SDK:

- Session: session number to use for inventory operation
- Tag population: an estimate of the tag population in view of the RF field of the antenna
- Select (SL flag)
- Target (inventory state).

Singulation control settings could be retrieved and set via accordingly *srfidGetSingulationConfiguration* and *srfidSetSingulationConfiguration* API functions as demonstrated in the following example.

```
/* identifier of one of active RFID readers is supposed to be stored in m_ReaderId variable */

/* allocate object for storage of singulation settings */
srfidSingulationConfig *singulation_cfg = [[srfidSingulationConfig alloc] init];

/* an object for storage of error response received from RFID reader */
NSString *error_response = nil;

/* retrieve singulation configuration */
SRFID_RESULT result = [apiInstance srfidGetSingulationConfiguration:m_ReaderId
aSingulationConfig:&singulation_cfg   aStatusMessage:&error_response];
if (SRFID_RESULT_SUCCESS == result) {
    /* singulation configuration received */
    NSLog(@"Tag population: %d\n", [singulation_cfg getTagPopulation]);
```

(continued on next page)

```
    SRFID_SLFLAG slflag = [singulation_cfg getSLFlag];
    switch (slflag) {
        case SRFID_SLFLAG_ASSERTED:
            NSLog(@"SL flag: ASSERTED\n");
            break;
    case SRFID_SLFLAG_DEASSERTED:
        NSLog(@"SL flag: DEASSERTED\n");
        break;
    case SRFID_SLFLAG_ALL:
        NSLog(@"SL flag: ALL\n");
        break;
    }

    SRFID_SESSION session = [singulation_cfg getSession];
    switch (session) {
        case SRFID_SESSION_S1:
            NSLog(@"Session: S1\n");
            break;
        case SRFID_SESSION_S2:
            NSLog(@"Session: S2\n");
            break;
        case SRFID_SESSION_S3:
            NSLog(@"Session: S3\n");
            break;
        case SRFID_SESSION_S0:
            NSLog(@"Session: S0\n");
            break;
    }
    SRFID_INVENTORYSTATE state = [singulation_cfg getInventoryState];
    switch (state) {
        case SRFID_INVENTORYSTATE_A:
            NSLog(@"Inventory State: State A\n");
            break;
        case SRFID_INVENTORYSTATE_B:
            NSLog(@"Inventory State: State B\n");
            break;
            case SRFID_INVENTORYSTATE_AB_FLIP:
            NSLog(@"Inventory State: AB flip\n");
            break;
    }
    /* change the received singulation configuration */
    [singulation_cfg setTagPopulation:30];
    [singulation_cfg setSession:SRFID_SESSION_S0];
    [singulation_cfg setSlFlag:SRFID_SLFLAG_ASSERTED];
    [singulation_cfg setInventoryState:SRFID_INVENTORYSTATE_A];

    error_response = nil;

    /* set updated singulation configuration */
    result = [apiInstance srfidSetSingulationConfiguration:m_ReaderId
aSingulationConfig:singulation_cfg aStatusMessage:&error_response];
```

(continued on next page)

```
    if (SRFID_RESULT_SUCCESS == result) {
        /* singulation configuration applied successfully */
        NSLog(@"Singulation configuration has been set\n");
    }
    else if (SRFID_RESULT_RESPONSE_ERROR == result) {
        NSLog(@"Error response from RFID reader: %@\n", error_response);
    }
    else if (SRFID_RESULT_RESPONSE_TIMEOUT == result) {
        NSLog(@"Timeout occurs during communication with RFID reader\n");
    }
    else if (SRFID_RESULT_READER_NOT_AVAILABLE == result) {
        NSLog(@"RFID reader with id = %d is not available\n", m_ReaderId);
    }
    else {
        NSLog(@"Request failed\n");
    }
}
else if (SRFID_RESULT_RESPONSE_ERROR == result) {
    NSLog(@"Error response from RFID reader: %@\n", error_response);
}
else if (SRFID_RESULT_RESPONSE_TIMEOUT == result) {
    NSLog(@"Timeout occurs during communication with RFID reader\n");
}
else if (SRFID_RESULT_READER_NOT_AVAILABLE == result) {
    NSLog(@"RFID reader with id = %d is not available\n", m_ReaderId);
}
else {
    NSLog(@"Request failed\n");
}

[singulation_cfg release];
```

Trigger Configuration

The SDK provides an ability to configure start and stop trigger parameters. Start trigger parameters include the following:

- Start of an operation based on a physical trigger.
- Trigger type (press/release) of a physical trigger.
- Delay (in milliseconds) of start of operation.
- Repeat monitoring for start trigger after stop of operation.

Start trigger configuration could be retrieved and set via *srfidGetStartTriggerConfiguration* and *srfidSetStartTriggerConfiguration* API functions accordingly.

Stop trigger parameters include the following:

- Stop of an operation based on a physical trigger.
- Trigger type (press/release) of a physical trigger.
- Stop of an operation based on a specified number of tags inventoried.
- Stop of an operation based on a specified timeout (in milliseconds).
- Stop of an operation based on a specified number of inventory rounds completed.
- Stop of an operation based on a specified number of access rounds completed.

Stop trigger settings could be retrieved and set via accordingly *srfidGetStopTriggerConfiguration* and *srfidSetStopTriggerConfiguration* API functions.

The following example demonstrates retrieval of current start and stop trigger parameters as well as configuring new start and stop triggers parameters.

```
/* identifier of one of active RFID readers is supposed to be stored in m_ReaderId variable */

/* allocate object for storage of start trigger settings */
srfidStartTriggerConfig *start_trigger_cfg = [[srfidStartTriggerConfig alloc] init];

/* an object for storage of error response received from RFID reader */
NSString *error_response = nil;

/* retrieve start trigger parameters */
SRFID_RESULT result = [apiInstance srfidGetStartTriggerConfiguration:m_ReaderId
aStartTriggeConfig:&start_trigger_cfg   aStatusMessage:&error_response];

if (SRFID_RESULT_SUCCESS == result) {
     /* start trigger configuration received */
     NSLog(@"Start trigger: start on physical trigger = %@\n", ((YES == [start_trigger_cfg
getStartOnHandheldTrigger]) ? @"YES" : @"NO"));
     NSLog(@"Start trigger: physical trigger type = %@\n",
((SRFID_TRIGGERTYPE_PRESS == [start_trigger_cfg getTriggerType]) ? @"PRESSED" :
@"RELEASED"));
     NSLog(@"Start trigger: delay = %d ms\n", [start_trigger_cfg getStartDelay]);
     NSLog(@"Start trigger: repeat monitoring = %@\n", ((NO == [start_trigger_cfg
getRepeatMonitoring]) ? @"NO" : @"YES"));
}
else {

     NSLog(@"Failed to receive start trigger parameters\n");
}

/* configure start trigger parameters */
/* start on physical trigger */
[start_trigger_cfg setStartOnHandheldTrigger:YES];
/* start on physical trigger press */
[start_trigger_cfg setTriggerType:SRFID_TRIGGERTYPE_PRESS];
/* repeat monitoring for start trigger conditions after operation stop */
[start_trigger_cfg setRepeatMonitoring:YES];
[start_trigger_cfg setStartDelay:0];
/* set start trigger parameters */
result = [apiInstance srfidSetStartTriggerConfiguration:m_ReaderId
aStartTriggeConfig:start_trigger_cfg   aStatusMessage:&error_response];
if (SRFID_RESULT_SUCCESS == result) {
     /* start trigger configuration applied */
     NSLog(@"Start trigger configuration has been set\n");
}
else {
     NSLog(@"Failed to set start trigger parameters\n");
}
[start_trigger_cfg release];

/* allocate object for storage of start trigger settings */
srfidStopTriggerConfig *stop_trigger_cfg = [[srfidStopTriggerConfig alloc] init];
```

(continued on next page)

```
/* retrieve stop trigger parameters */
result = [apiInstance srfidGetStopTriggerConfiguration:m_ReaderId
aStopTriggeConfig:&stop_trigger_cfg   aStatusMessage:&error_response];

if (SRFID_RESULT_SUCCESS == result) {
     /* stop trigger configuration received */
     NSLog(@"Stop trigger: start on physical trigger = %@\n", ((YES ==
[stop_trigger_cfg getStopOnHandheldTrigger]) ? @"YES" : @"NO"));
     NSLog(@"Stop trigger: physical trigger type = %@\n",
((SRFID_TRIGGERTYPE_PRESS == [stop_trigger_cfg getTriggerType]) ? @"PRESSED" :
@"RELEASED"));
         if (YES == [stop_trigger_cfg getStopOnTagCount]) {
            NSLog(@"Stop trigger: stop on %d number of tags received\n",
[stop_trigger_cfg getStopTagCount]);
         }
         if (YES == [stop_trigger_cfg getStopOnTimeout]) {
            NSLog(@"Stop trigger: stop on %d ms timeout\n", [stop_trigger_cfg
getStopTimeout]);
     }
         if (YES == [stop_trigger_cfg getStopOnInventoryCount]) {
            NSLog(@"Stop trigger: stop on %d inventory rounds\n", [stop_trigger_cfg
getStopInventoryCount]);
         }
         if (YES == [stop_trigger_cfg getStopOnAccessCount]) {
            NSLog(@"Stop trigger: stop on %d access rounds\n", [stop_trigger_cfg
getStopAccessCount]);
     }
}
else {
     NSLog(@"Failed to receive stop trigger parameters\n");
}
/* configure stop trigger parameters: stop on physical trigger release or after 5 sec
timeout or after 10 tags inventoried */
/* start on physical trigger */
[stop_trigger_cfg setStopOnHandheldTrigger:YES];
[stop_trigger_cfg setTriggerType:SRFID_TRIGGERTYPE_RELEASE];
[stop_trigger_cfg setStopOnTimeout:YES];
[stop_trigger_cfg setStopTimout:(5*1000)];
[stop_trigger_cfg setStopOnTagCount:YES];
[stop_trigger_cfg setStopTagCount:10];
[stop_trigger_cfg setStopOnInventoryCount:NO];
[stop_trigger_cfg setStopOnAccessCount:NO];

/* set stop trigger parameters */
result = [apiInstance srfidSetStopTriggerConfiguration:m_ReaderId
aStopTriggeConfig:stop_trigger_cfg aStatusMessage:&error_response];
if (SRFID_RESULT_SUCCESS == result) {
     /* stop trigger configuration applied */
     NSLog(@"Stop trigger configuration has been set\n");
}
else {
     NSLog(@"Failed to set stop trigger parameters\n");
}
[stop_trigger_cfg release];
```

Tag Report Configuration

The SDK provides an ability to configure a set of fields to be reported in a response to an operation by a specific active RFID reader.

Supported fields that might be reported include the following:

- First and last seen times
- PC value
- RSSI value
- Phase value
- Channel index
- Tag seen count.

Tag report parameters could be managed via *srfidSetReportConfiguration* and *srfidGetReportConfiguration* API functions as demonstrated in the following example.

```
/* identifier of one of active RFID readers is supposed to be stored in m_ReaderId variable */

/* allocate object for storage of tag report settings */
srfidTagReportConfig *report_cfg = [[srfidTagReportConfig alloc] init];

/* an object for storage of error response received from RFID reader */
NSString *error_response = nil;

/* retrieve tag report parameters */
SRFID_RESULT result = [apiInstance srfidGetTagReportConfiguration:m_ReaderId
aTagReportConfig:&report_cfg aStatusMessage:&error_response];
if (SRFID_RESULT_SUCCESS == result) {
     /* tag report configuration received */
     NSLog(@"PC field: %@\n", ((NO == [report_cfg getIncPC]) ? @"off" : @"on"));
     NSLog(@"Phase field: %@\n", ((NO == [report_cfg getIncPhase]) ? @"off" :
@"on"));
     NSLog(@"Channel index field: %@\n", ((NO == [report_cfg getIncChannelIdx]) ?
@"off" : @"on"));
     NSLog(@"RSSI field: %@\n", ((NO == [report_cfg getIncRSSI]) ? @"off" :
@"on"));
     NSLog(@"Tag seen count field: %@\n", ((NO == [report_cfg getIncTagSeenCount])
? @"off" : @"on"));
     NSLog(@"First seen time field: %@\n", ((NO == [report_cfg getIncFirstSeenTime]) ?
@"off" : @"on"));
     NSLog(@"Last seen time field: %@\n", ((NO == [report_cfg getIncLastSeenTime])
? @"off" : @"on"));
}
else {
     NSLog(@"Failed to receive tag report parameters\n");
}

/* configure tag report parameters to include only RSSI field */
[report_cfg setIncRSSI:YES];
[report_cfg setIncPC:NO]; [report_cfg setIncPhase:NO]; [report_cfg setIncChannelIdx:NO];
[report_cfg setIncTagSeenCount:NO]; [report_cfg setIncFirstSeenTime:NO]; [report_cfg
setIncLastSeenTime:NO];
```

(continued on next page)

```
/* set tag report parameters */
result = [apiInstance srfidSetTagReportConfiguration:m_ReaderId
aTagReportConfig:report_cfg aStatusMessage:&error_response];
if (SRFID_RESULT_SUCCESS == result) {
     /* tag report configuration applied */
     NSLog(@"Tag report configuration has been set\n");
}
else {
     NSLog(@"Failed to set tag report parameters\n");
}
[report_cfg release];
```

Regulatory Configuration

The SDK supports managing of regulatory related parameters of a specific active RFID reader.

Regulatory configuration includes the following:

- Code of selected region
- Hopping
- Set of enabled channels.

A set of enabled channels includes only such channels that are supported in the selected region. If hopping configuration is not allowed for the selected regions a set of enabled channels is not specified.

Regulatory parameters could be retrieved and set via *srfidGetRegulatoryConfig* and *srfidSetRegulatoryConfig* API functions accordingly. The following example demonstrates retrieving of current regulatory settings and configuring the RFID reader to operate in one of supported regions.

```
/* identifier of one of active RFID readers is supposed to be stored in m_ReaderId variable */
/* allocate object for storage of regulatory settings */
srfidRegulatoryConfig *regulatory_cfg = [[srfidRegulatoryConfig alloc] init];

/* an object for storage of error response received from RFID reader */
NSString *error_response = nil;

/* retrieve regulatory parameters */
SRFID_RESULT result = [apiInstance srfidGetRegulatoryConfig:m_ReaderId
aRegulatoryConfig:&regulatory_cfg aStatusMessage:&error_response];
if (SRFID_RESULT_SUCCESS == result) {
     /* regulatory configuration received */
     if (NSOrderedSame == [[regulatory_cfg getRegionCode] caseInsensitiveCompare:@"NA"]) {
         NSLog(@"Regulatory: region is NOT set\n");
     }
     else {
         NSLog(@"Region code: %@\n", [regulatory_cfg getRegionCode]);
         SRFID_HOPPINGCONFIG hopping_cfg = [regulatory_cfg getHoppingConfig];
         NSLog(@"Hopping is %@\n", ((SRFID_HOPPINGCONFIG_DISABLED == hopping_cfg)
? @"off" : @"on"));
         NSArray *channels = [regulatory_cfg getEnabledChannelsList];
         for (NSString *str in channels) {
            NSLog(@"Enabled channel: %@\n", str);
         }
     }
}
else {
```

(continued on next page)

```
     NSLog(@"Failed to receive regulatory parameters\n");
}

[regulatory_cfg release];

/* code of region to be set as current one */
NSString *region_code = @"USA";
/* an array of enabled channels to be set */
NSMutableArray *enabled_channels = [[NSMutableArray alloc] init];
/* a hopping to be set */
SRFID_HOPPINGCONFIG hopping_on = SRFID_HOPPINGCONFIG_DISABLED;

/* allocate object for storage of region information */
NSMutableArray *regions = [[NSMutableArray alloc] init];

/* retrieve supported regions */
result = [apiInstance srfidGetSupportedRegions:m_ReaderId aSupportedRegions:&regions
aStatusMessage:&error_response];

if (SRFID_RESULT_SUCCESS == result) {
     /* supported regions information received */
     /* select the last supported regions to be set as current one */
     region_code = [NSString stringWithFormat:@"%@", [(srfidRegionInfo*)[regions
lastObject] getRegionCode]];
}

[regions release];
/* allocate object for storage of supported channels information */
NSMutableArray *supported_channels = [[NSMutableArray alloc] init]; BOOL
hopping_configurable = NO;

/* retrieve detailed information about region specified by region code */
result = [apiInstance srfidGetRegionInfo:m_ReaderId aRegionCode:region_code
aSupportedChannels:&supported_channels    aHoppingConfigurable:&hopping_configurable
aStatusMessage:&error_response];

if (SRFID_RESULT_SUCCESS == result) {
     /* region information received */

     if (YES == hopping_configurable) {
         /* region supports hopping */
         /* enable first and last channels from the set of supported channels */
         [enabled_channels addObject:[supported_channels firstObject]];
         [enabled_channels addObject:[supported_channels lastObject]];
         /* enable hopping */
         hopping_on = SRFID_HOPPINGCONFIG_ENABLED;
     }
     else {
         /* region does not support hopping */
         /* request to not configure hopping */
         hopping_on = SRFID_HOPPINGCONFIG_DEFAULT;
     }
}
```

(continued on next page)

```
[supported_channels release]; error_response = nil;

/* configure regulatory parameters to be set */
regulatory_cfg = [[srfidRegulatoryConfig alloc] init]; [regulatory_cfg
setRegionCode:region_code]; [regulatory_cfg setEnabledChannelsList:enabled_channels];
[regulatory_cfg setHoppingConfig:hopping_on];

/* set regulatory parameters */
result = [apiInstance srfidSetRegulatoryConfig:m_ReaderId
aRegulatoryConfig:regulatory_cfg aStatusMessage:&error_response];
if (SRFID_RESULT_SUCCESS == result) {
     /* regulatory configuration applied */
     NSLog(@"Tag report configuration has been set\n");
}
else if (SRFID_RESULT_RESPONSE_ERROR == result) {
     NSLog(@"Error response from RFID reader: %@\n", error_response);
}
else {
NSLog(@"Failed to set regulatory parameters\n");
}
[enabled_channels release]; [regulatory_cfg release];
```

Pre-filters Configuration

Pre-filters are same as the select command of C1G2 specification. The SDK supports pr e-filters configuration of a specific active RFID reader. When pre -filters are configured, they could be applied prior to inventory operations.

Following parameters could be configured for each pre -filter:

- Target (Session S0, Session S1, Session S2, Se ssion S3, Select Flag)
- Action
- Memory bank (EPC, TID, USER)
- Mask start position (in words): indicates start position from beginning of memory bank from were match pattern is checked
- Match pattern.

Configured pre-filters could be retrieved from a specific active RFID reader via *srfidGetPreFilters* API function. The *srfidSetPreFilters* API function is used to configure a new set of pre -filters. The following example demonstrates pre-filters management supported by the SDK.

```
/* identifier of one of active RFID readers is supposed to be stored in m_ReaderId variable */

/* allocate object for storage of pre filters */
NSMutableArray *prefilters = [[NSMutableArray alloc] init];
/* an object for storage of error response received from RFID reader */
NSString *error_response = nil;
/* retrieve pre-filters */
SRFID_RESULT result = [apiInstance srfidGetPreFilters:m_ReaderId
aPreFilters:&prefilters aStatusMessage:&error_response];
if (SRFID_RESULT_SUCCESS == result) {
     /* pre-filters received */
     NSLog(@"Number of pre-filters: %d\n", [prefilters count]);
```

(continued on next page)

```
for (srfidPreFilter *filter in prefilters) {
    NSLog(@"Match pattern: %@\n", [filter getMatchPattern]);
    NSLog(@"Mask start position: %d words\n", [filter getMaskStartPos]);

    SRFID_SELECTACTION action = [filter getAction];
    switch (action) {
        case SRFID_SELECTACTION_INV_A2BB2A_NOT_INV_A OR NEG_SL_NOT_ASRT_SL:
            NSLog(@"Action: INV A2BB2A NOT INV A OR NEG SL NOT ASRT SL\n");
            break;
        case SRFID_SELECTACTION_INV_AOR_ASRT_SL:
            NSLog(@"Action: INV A OR ASRT SL\n");
            break;
        case SRFID_SELECTACTION_INV_A_NOT_INV_BOR_ASRT_SL_NOT_DSRT_SL:
            NSLog(@"Action: INV A NOT INV B OR ASRT SL NOT DSRT SL\n");
            break;
        case SRFID_SELECTACTION_INV_BOR_DSRT_SL:
            NSLog(@"Action: INV B OR DSRT SL\n");
            break;
        case SRFID_SELECTACTION_INV_B_NOT_INV_AOR_DSRT_SL_NOT_ASRT_SL:
            NSLog(@"Action: INV B NOT INV A OR DSRT SL NOT ASRT SL\n");
            break;
        case SRFID_SELECTACTION_NOT_INV_A2BB2AOR_NOT_NEG_SL:
            NSLog(@"Action: NOT INV A2BB2A OR NOT NEG SL\n");
            break;
        case SRFID_SELECTACTION_NOT_INV_AOR_NOT_ASRT_SL:
            NSLog(@"Action: NOT INV A OR NOT ASRT SL\n");
            break;
        case SRFID_SELECTACTION_NOT_INV_BOR_NOT_DSRT_SL:
            NSLog(@"Action: NOT INV B OR NOT DSRT SL\n");
            break;
    }
    SRFID_SELECTTARGET target = [filter getTarget];
    switch (target) {
        case SRFID_SELECTTARGET_S0:
            NSLog(@"Target: Session SO\n");
            break;
        case SRFID_SELECTTARGET_S1:
            NSLog(@"Target: Session S1\n"); break;
        case SRFID_SELECTTARGET_S2:
            NSLog(@"Target: Session S2\n"); break;
        case SRFID_SELECTTARGET_S3:
            NSLog(@"Target: Session S3\n");
            break;
        case SRFID_SELECTTARGET_SL:
            NSLog(@"Target: Select Flag\n");
            break;
    }
```

(continued on next page)

```
            SRFID_MEMORYBANK bank = [filter getMemoryBank]; switch (bank) {
               case SRFID_MEMORYBANK_EPC:
                  NSLog(@"Memory Bank: EPC\n");
                  break;
               case SRFID_MEMORYBANK_RESV:
                  NSLog(@"Memory Bank: RESV\n");
                  break;
               case SRFID_MEMORYBANK_TID:
                  NSLog(@"Memory Bank: TID\n");
                  break;
               case SRFID_MEMORYBANK_USER:
                  NSLog(@"Memory Bank: USER\n");
                  break;
            }
         }
}

else {
     NSLog(@"Failed to receive pre-filters\n");
}
[prefilters removeAllObjects];
/* create one pre-filter */
srfidPreFilter *filter = [[srfidPreFilter alloc] init]; [filter
setMatchPattern:@"N20122014R1010364989126V"]; [filter setMaskStartPos:2];
[filter setMemoryBank:SRFID_MEMORYBANK_EPC];
[filter setAction:SRFID_SELECTACTION_INV_AOR
[filter setTarget:SRFID_SELECTTARGET_SL];
[prefilters addObject:filter]; [filter release];
error_response = nil;
ASRT_SL];
/* set pre-filters */

result = [apiInstance srfidSetPreFilters:m_ReaderId aPreFilters:prefilters
aStatusMessage:&error_response];
if (SRFID_RESULT_SUCCESS == result) {
     /* pre-filters have been set */
     NSLog(@"Pre-filters has been set\n");
}
else if (SRFID_RESULT_RESPONSE_ERROR == result) {
     NSLog(@"Error response from RFID reader: %@\n", error_response);
}
else {
     NSLog(@"Failed to set tag report parameters\n");
}
[prefilters release];
```

Beeper Configuration

The SDK provides an ability to configure a beeper of a specific active RFID reader. The beeper could be configured to one of predefined volumes (low, medium, high) or be disabled. Retrieving and setting of beeper configuration is performed via *srfidSetBeeperConfig* and *srfidGetBeeperConfig* API functions as demonstrated in the following example.

```
/* identifier of one of active RFID readers is supposed to be stored in m_ReaderId variable */

/* object for beeper configuration */
SRFID_BEEPERCONFIG beeper_cfg;
/* an object for storage of error response received from RFID reader */
NSString *error_response = nil;
/* retrieve beeper configuration */
SRFID_RESULT result = [apiInstance srfidGetBeeperConfig:m_ReaderId
aBeeperConfig:&beeper_cfg aStatusMessage:&error_response];
if (SRFID_RESULT_SUCCESS == result) {
     /* beeper configuration received */
     switch (beeper_cfg) {
         case SRFID_BEEPERCONFIG_HIGH:
            NSLog(@"Beeper: high volume\n");
            break;
         case SRFID_BEEPERCONFIG_LOW:
            NSLog(@"Beeper: low volume\n");
            break;
         case SRFID_BEEPERCONFIG_MEDIUM:
            NSLog(@"Beeper: medium volume\n");
            break;
         case SRFID_BEEPERCONFIG_QUIET:
            NSLog(@"Beeper: disabled\n");
            break;
     }
}
else {
     NSLog(@"Failed to receive beeper parameters\n");

}

error_response = nil;

/* disable beeper */
result = [apiInstance srfidSetBeeperConfig:m_ReaderId
aBeeperConfig:SRFID_BEEPERCONFIG_QUIET   aStatusMessage:&error_response];
if (SRFID_RESULT_SUCCESS == result) {
     /* beeper configuration applied */
     NSLog(@"Beeper configuration has been set\n");
}
else if (SRFID_RESULT_RESPONSE_ERROR == result) {
NSLog(@"Error response from RFID reader: %@\n", error_response);
}
else {
     NSLog(@"Failed to set beeper configuration\n");
}
```

Managing Configuration

Various parameter of a specific RFID reader configured via SDK are lost after next power down. The SDK provides an ability to store and restore a persistent configuration of RFID reader. The srfidSaveReaderConfiguration API function could be used to either make current configuration persistent over power down and power up cycles or store current configuration to custom defaults area. The configuration stored to custom defaults area could be restored via srfidRestoreReaderConfiguration API function. The same API function is used to restore the factory defined configuration.

The following example demonstrates utilization of mentioned API functions.

```
/* identifier of one of active RFID readers is supposed to be stored in m_ReaderId variable */
/* an object for storage of error response received from RFID reader */
NSString *error_response = nil;
/* cause the RFID reader to make current configuration persistent */
SRFID_RESULT result = [apiInstance srfidSaveReaderConfiguration:m_ReaderId
aSaveCustomDefaults:NO aStatusMessage:&error_response];
if (SRFID_RESULT_SUCCESS == result) {
     NSLog(@"Current configuration became persistent\n");
}
else {
     NSLog(@"Request failed\n");
}
/* cause the RFID reader to save current configuration in custom defaults area */
result = [apiInstance srfidSaveReaderConfiguration:m_ReaderId aSaveCustomDefaults:YES
aStatusMessage:&error_response];
if (SRFID_RESULT_SUCCESS == result) {
     NSLog(@"Current configuration stored in custom defaults\n");
}
else {
     NSLog(@"Request failed\n");
}
/* cause the RFID reader to restore configuration from custom defaults */
result = [apiInstance srfidRestoreReaderConfiguration:m_ReaderId
aRestoreFactoryDefaults:NO aStatusMessage:&error_response];
if (SRFID_RESULT_SUCCESS != result) { NSLog(@"Request failed\n");
}

/* cause the RFID reader to restore factory defined configuration*/
result = [apiInstance srfidRestoreReaderConfiguration:m_ReaderId
aRestoreFactoryDefaults:YES aStatusMessage:&error_response];
if (SRFID_RESULT_SUCCESS != result) {
     NSLog(@"Request failed\n");
}
```

Performing Operations

The Zebra RFID SDK for iOS API enables performing various radio operations with a specific active RFID reader.

Rapid Read

Rapid read operation is a simple inventory operation wi thout performing a read from a specific memory bank.

The *srfidStartRapidRead* API function is used to request performing of rapid read operation. Aborting of rapid read operation is requested via *srfidStopRapidRead* API function. When performing of rapid read operation is requested the actual operation will be started once conditions specified by start trigger parameters are met. The on-going operation will be stopped in accordance with configured stop trigger parameters. If repeat monitoring option is ena bled in start trigger configuration the actual operation will be started again after it has stopped once conditions of start trigger configuration are met. On starting and stopping of the actual operation the SDK will deliver asynchronous notifications to the application if the application has subscribed for events of this type.

The SDK will deliver asynchronous notifications to inform the application about tag data received from the RFID reader during the on-going operation if the application has subscribe d for events of this type. Fields to be reported during asynchronous tag data related notification are configured via *reportConfig* parameter of *srfidStartRapidRead* API function.

The following example demonstrates performing of rapid read operation that starts and stops immediately after requested operation performing and aborting.

```
/* subscribe for tag data related events */
[apiInstance srfidSubsribeForEvents:SRFID_EVENT_MASK_READ];
/* subscribe for operation start/stop related events */
[apiInstance srfidSubsribeForEvents:SRFID_EVENT_MASK_STATUS];
/* identifier of one of active RFID readers is supposed to be stored in m_ReaderId variable */
/* allocate object for start trigger settings */
srfidStartTriggerConfig *start_trigger_cfg = [[srfidStartTriggerConfig alloc]
init];
/* allocate object for stop trigger settings */
srfidStopTriggerConfig *stop_trigger_cfg = [[srfidStopTriggerConfig alloc] init];
/* allocate object for report parameters of rapid read operation */
srfidReportConfig *report_cfg = [[srfidReportConfig alloc] init];

/* allocate object for access parameters of rapid read operation */
srfidAccessConfig *access_cfg = [[srfidAccessConfig alloc] init];

/* an object for storage of error response received from RFID reader */
NSString *error_response = nil;

do {
     /* configure start and stop triggers parameters to start and stop actual
operation immediately on a corresponding response */
     [start_trigger_cfg setStartOnHandheldTrigger:NO];
     [start_trigger_cfg setStartDelay:0];
     [start_trigger_cfg setRepeatMonitoring:NO];
```

(continued on next page)

```
    [stop_trigger_cfg setStopOnHandheldTrigger:NO];
    [stop_trigger_cfg setStopOnTimeout:NO];
    [stop_trigger_cfg setStopOnTagCount:NO];
    [stop_trigger_cfg setStopOnInventoryCount:NO];
    [stop_trigger_cfg setStopOnAccessCount:NO];

    /* set start trigger parameters */
    SRFID_RESULT result = [apiInstance
srfidSetStartTriggerConfiguration:m_ReaderId aStartTriggeConfig:start_trigger_cfg
aStatusMessage:&error_response];
    if (SRFID_RESULT_SUCCESS == result) {
        /* start trigger configuration applied */
        NSLog(@"Start trigger configuration has been set\n");
    }
    else {
        NSLog(@"Failed to set start trigger parameters\n");
        break;
    }

    /* set stop trigger parameters */
    result = [apiInstance srfidSetStopTriggerConfiguration:m_ReaderId
aStopTriggeConfig:stop_trigger_cfg aStatusMessage:&error_response];
    if (SRFID_RESULT_SUCCESS == result) {
        /* stop trigger configuration applied */
        NSLog(@"Stop trigger configuration has been set\n");
    }
    else {
        NSLog(@"Failed to set stop trigger parameters\n"); break;
    }

    /* start and stop triggers have been configured */
    error_response = nil;

    /* configure report parameters to report RSSI, Channel Index, Phase and PC fields */
        [report_cfg setIncPC:YES];
        [report_cfg setIncPhase:YES];
        [report_cfg setIncChannelIndex:YES];
        [report_cfg setIncRSSI:YES];
        [report_cfg setIncTagSeenCount:NO];
        [report_cfg setIncFirstSeenTime:NO];
        [report_cfg setIncLastSeenTime:NO];

        /* configure access parameters to perform the operation with 27.0 dbm antenna
power level without application of pre-filters */
        [access_cfg setPower:270];
        [access_cfg setDoSelect:NO];

    /* request performing of rapid read operation */
    result = [apiInstance srfidStartRapidRead:m_ReaderId aReportConfig:report_cfg
aAccessConfig:access_cfg aStatusMessage:&error_response];
    if (SRFID_RESULT_SUCCESS == result) {
        NSLog(@"Request succeed\n");
```

(continued on next page)

```
        /* stop an operation after 1 minute */
        dispatch_after(dispatch_time(DISPATCH_TIME_NOW, (int64_t)(60 *
NSEC_PER_SEC)), dispatch_get_main_queue(), ^{
            [apiInstance srfidStopRapidRead:m_ReaderId aStatusMessage:nil];
        });
    }
    else if (SRFID_RESULT_RESPONSE_ERROR == result) {
        NSLog(@"Error response from RFID reader: %@\n", error_response);
    }
    else {
        NSLog(@ÒRequest failed\nÓ);
    }
} while (0); [start_trigger_cfg release];
[stop_trigger_cfg release];
[access_cfg release];
[report_cfg release];
/* EventReceiver class: partial implementation */
@implementation EventReceiver
...

-    (void)srfidEventReadNotify:(int)readerID aTagData:(srfidTagData*)tagData {
/* print the received tag data */
NSLog(@"Tag data received from RFID reader with ID = %d\n", readerID); NSLog(@"Tag id:
%@\n", [tagData getTagId]);
}

-    (void)srfidEventStatusNotify:(int)readerID aEvent:(SRFID_EVENT_STATUS)event {
NSLog(@"Radio operation has %@\n", ((SRFID_EVENT_STATUS_OPERATION_START ==
event) ? @"started" : @"stopped"));
}
...
@end
```

Inventory

Inventory is an advanced inventory operation being performed simultaneously with reading from a specific memory bank.

Inventory operation is performed similarly to the rapid read operation described above. Thus performing and aborting of the inventory operation is requested through *srfidStartInventory* and *srfidStopInventory* API functions accordingly. After request of o peration performing the actual operation will be started in accordance with the configured start trigger parameters and will be stopped once conditions specified by stop trigger parameters are met. After the operation has stopped it might be started again if it is not aborted and repeat monitoring option is enabled in start trigger configuration. The SDK informs the application about starting and stopping of the actual notification trough corresponding asynchronous notifications.

The SDK will deliver asynchronous notifications to inform the application about tag data received from the RFID reader during the on-going operation if the application has subscribed for events of this type. Fields to be reported during asynchronous tag data related notification are configured via reportConfig parameter of srfidStartInventory API function.

The following example demonstrates performing of a continuous inventory operation with reading from EPC memory bank that starts on a press of a physical trigger and stops on a release of a physical trigger or after a 25 second timeout.

(continued on next page)

```
/* subscribe for tag data related events */
[apiInstance srfidSubsribeForEvents:SRFID_EVENT_MASK_READ];
/* subscribe for operation start/stop related events */
[apiInstance srfidSubsribeForEvents:SRFID_EVENT_MASK_STATUS];
/* identifier of one of active RFID readers is supposed to be stored in m_ReaderId variable */
/* allocate object for start trigger settings */
srfidStartTriggerConfig *start_trigger_cfg = [[srfidStartTriggerConfig alloc]
init];
/* allocate object for stop trigger settings */
srfidStopTriggerConfig *stop_trigger_cfg = [[srfidStopTriggerConfig alloc] init];

/* allocate object for report parameters of inventory operation */
srfidReportConfig *report_cfg = [[srfidReportConfig alloc] init];

/* allocate object for access parameters of inventory operation */
srfidAccessConfig *access_cfg = [[srfidAccessConfig alloc] init];

/* an object for storage of error response received from RFID reader */
NSString *error_response = nil;

do {
     /* configure start triggers parameters to start on physical trigger press */
     [start_trigger_cfg setStartOnHandheldTrigger:YES];
     [start_trigger_cfg setTriggerType:SRFID_TRIGGERTYPE_PRESS];
     [start_trigger_cfg setStartDelay:0];
     [start_trigger_cfg setRepeatMonitoring:YES];
     /* configure stop triggers parameters to stop on physical trigger release or on 25 sec
timeout*/
     [stop_trigger_cfg setStopOnHandheldTrigger:YES];
     [stop_trigger_cfg setTriggerType:SRFID_TRIGGERTYPE_RELEASE];
     [stop_trigger_cfg setStopOnTimeout:YES];
     [stop_trigger_cfg setStopTimout:(25*1000)];
     [stop_trigger_cfg setStopOnTagCount:NO];
     [stop_trigger_cfg setStopOnInventoryCount:NO];
     [stop_trigger_cfg setStopOnAccessCount:NO];

     /* set start trigger parameters */
     SRFID_RESULT result = [apiInstance
srfidSetStartTriggerConfiguration:m_ReaderId    aStartTriggeConfig:start_trigger_cfg
aStatusMessage:&error_response];
     if (SRFID_RESULT_SUCCESS == result) {
         /* start trigger configuration applied */
         NSLog(@"Start trigger configuration has been set\n");
     }
     else {
         NSLog(@"Failed to set start trigger parameters\n");
         break;
     }

     /* set stop trigger parameters */
     result = [apiInstance srfidSetStopTriggerConfiguration:m_ReaderId
aStopTriggeConfig:stop_trigger_cfg aStatusMessage:&error_response];
     if (SRFID_RESULT_SUCCESS == result) {
         /* stop trigger configuration applied */
         NSLog(@"Stop trigger configuration has been set\n");
     }
```

(continued on next page)

```
    else {
        NSLog(@"Failed to set stop trigger parameters\n");
        break;
    }

    /* start and stop triggers have been configured */
error_response = nil;

    /* configure report parameters to report RSSI and Channel Index fields */
    [report_cfg setIncPC:NO];
    [report_cfg setIncPhase:NO];
    [report_cfg setIncChannelIndex:YES];
    [report_cfg setIncRSSI:YES]; [report_cfg setIncTagSeenCount:NO];
    [report_cfg setIncFirstSeenTime:NO];
    [report_cfg setIncLastSeenTime:NO];

    /* configure access parameters to perform the operation with 27.0 dbm antenna power
level without application of pre-filters */
    [access_cfg setPower:270];
    [access_cfg setDoSelect:NO];

    /* request performing of inventory operation with reading from EPC memory bank */
    result = [apiInstance srfidStartInventory:m_ReaderId
aMemoryBank:SRFID_MEMORYBANK_EPC aReportConfig:report_cfg
aAccessConfig:access_cfg aStatusMessage:&error_response];

    if (SRFID_RESULT_SUCCESS == result) {
        NSLog(@"Request succeed\n");
        /* request abort of an operation after 1 minute */
        dispatch_after(dispatch_time(DISPATCH_TIME_NOW, (int64_t)(60 *
NSEC_PER_SEC)), dispatch_get_main_queue(), ^{
        [apiInstance srfidStopInventory:m_ReaderId aStatusMessage:nil];
        });
    }
    else if (SRFID_RESULT_RESPONSE_ERROR == result) {
        NSLog(@"Error response from RFID reader: %@\n", error_response);
    }
    else {
        NSLog(@"Request failed\n");
    }
} while (0);
[start_trigger_cfg release];
[stop_trigger_cfg release];
[access_cfg release];
[report_cfg release];
/* EventReceiver class: partial implementation */
@implementation EventReceiver
...
-    (void)srfidEventStatusNotify:(int)readerID aEvent:(SRFID_EVENT_STATUS)event {
        NSLog(@"Radio operation has %@\n", ((SRFID_EVENT_STATUS_OPERATION_START ==
event) ? @"started" : @"stopped"));
}
- (void)srfidEventReadNotify:(int)readerID aTagData:(srfidTagData*)tagData {
```

(continued on next page)

```
    /* print the received tag data */
    NSLog(@"Tag data received from RFID reader with ID = %d\n", readerID);
    NSLog(@"Tag id: %@\n", [tagData getTagId]);
    SRFID_MEMORYBANK bank = [tagData getMemoryBank];
    if (SRFID_MEMORYBANK_NONE != bank) {
        NSString *str_bank = @"";
        switch (bank) {
            case SRFID_MEMORYBANK_EPC:
                str_bank = @"EPC";
                break;
            case SRFID_MEMORYBANK_TID:
                str_bank = @"TID";
                break;
            case SRFID_MEMORYBANK_USER:
                str_bank = @"USER";
                break;
            case SRFID_MEMORYBANK_RESV:
                str_bank = @"RESV";
                break;
        }
        NSLog(@"%@ memory bank data: %@\n", str_bank, [tagData getMemoryBankData]);
    }
}
...
@end
```

Inventory with Pre-filters

If pre-filters are configured they might be applied during performing of inventory operation. Application of pre-filters is enabled via *accessConfig* parameter of *srfidStartInventory* and *srfidStartRapidRead* API functions. Excepting enablement of pre-filters application in *accessConfig* parameter inventory with pre-filters is performed similarly to a typical inventory operation described above. The following example demonstrates enabling application of configured pre-filters during inventory operation.

```
/* allocate object for report parameters of inventory operation */
srfidReportConfig *report_cfg = [[srfidReportConfig alloc] init];
/* allocate object for access parameters of inventory operation */
srfidAccessConfig *access_cfg = [[srfidAccessConfig alloc] init];
/* an object for storage of error response received from RFID reader */
NSString *error_response = nil;
/* configure report parameters to report RSSI field */
[report_cfg setIncPC:NO];
[report_cfg setIncPhase:NO];
[report_cfg setIncChannelIndex:NO];
[report_cfg setIncRSSI:YES]; [report_cfg setIncTagSeenCount:NO]; [report_cfg
setIncFirstSeenTime:NO]; [report_cfg setIncLastSeenTime:NO];
/* configure access parameters to perform the operation with 27.0 dbm antenna power level */
[access_cfg setPower:270];
/* enable application of configured pre-filters */
[access_cfg setDoSelect:YES];
/* request performing of inventory operation with reading from EPC memory bank */
SRFID_RESULT result = [apiInstance srfidStartInventory:m_ReaderId
aMemoryBank:SRFID_MEMORYBANK_EPC aReportConfig:report_cfg aAccessConfig:access_cfg
aStatusMessage:&error_response];
```

(continued on next page)

```
if (SRFID_RESULT_SUCCESS == result)
     { NSLog(@"Request succeed\n");
     /* request abort of an operation after 1 minute */
     dispatch_after(dispatch_time(DISPATCH_TIME_NOW, (int64_t)(60 *
NSEC_PER_SEC)), dispatch_get_main_queue(), ^{
         [apiInstance srfidStopInventory:m_ReaderId aStatusMessage:nil];
     });
}
else if (SRFID_RESULT_RESPONSE_ERROR == result) {
     NSLog(@"Error response from RFID reader: %@\n", error_response);
}
else {
     NSLog(@"Request failed\n");
}
[access_cfg release];
[report_cfg release];
```

Tag Locationing

The SDK provides an ability to perform tag locationing operation. T he *srfidStartTagLocationing* API function is used to request performing of tag locationing operation. Aborting of tag locationing operation is requested via *srfidStopTagLocationing* API function. The actual operation is started and stopped based on configured start and stop triggers parameters. The SDK informs the application about starting and stopping of the actual operation via delivery of asynchronous notifications if the application has subscriber for events of this type. During an on -going operation the SDK will deliver asynchronous notifications to inform the application about current tag proximity value (in percents).

The following example demonstrates performing of tag locationing operation.

```
/* subscribe for tag locationing related events */
[apiInstance srfidSubsribeForEvents:SRFID_EVENT_MASK_PROXIMITY];
/* subscribe for operation start/stop related events */
[apiInstance srfidSubsribeForEvents:SRFID_EVENT_MASK_STATUS];
/* identifier of one of active RFID readers is supposed to be stored in m_ReaderId variable */

/* id of tag to be located */
NSString *tag_id = @"V6219894630101R41022102N";

/* an object for storage of error response received from RFID reader */
NSString *error_response = nil;

SRFID_RESULT result = [apiInstance srfidStartTagLocationing:m_ReaderId aTagEpcId:tag_id
aStatusMessage:&error_response];

if (SRFID_RESULT_SUCCESS == result) {
     NSLog(@"Request succeed\n");
     /* request abort of an operation after 1 minute */
     dispatch_after(dispatch_time(DISPATCH_TIME_NOW, (int64_t)(60 *
NSEC_PER_SEC)), dispatch_get_main_queue(), ^{
     [apiInstance srfidStopTagLocationing:m_ReaderId aStatusMessage:nil];
     });
}
else if (SRFID_RESULT_RESPONSE_ERROR == result) {
     NSLog(@"Error response from RFID reader: %@\n", error_response);
}
else {
     NSLog(@"Request failed\n");
}
```

(continued on next page)

```
/* EventReceiver class: partial implementation */
@implementation EventReceiver
...
-    (void)srfidEventStatusNotify:(int)readerID aEvent:(SRFID_EVENT_STATUS)event {
NSLog(@"Radio operation has %@\n", ((SRFID_EVENT_STATUS_OPERATION_START == event) ?
@"started" : @"stopped"));
}

- (void)srfidEventProximityNotify:(int)readerID aProximityPersent:(int)proximityPersent {
     NSLog(@"Tag proximity notification from RFID reader with ID = %d\n",
readerID);
     NSLog(@"Tag proximity: %d percents\n", proximityPersent);
}
...
@end
```

As shown in the following example, `TagLocationing` allows you to provide a mask to specify the contiguous bit pattern in the EPC ID that should be used to match the tag.

```
NSString *statusMessage;
NSString *testTagEpcId = @"111122223333444455556666";
NSString *tagMask = @"FFFFFFFF"
SRFID_RESULT result = [sdkInstance  srfidStartTagLocationing:m_rfidHelper.m_activeReaderId
aTagEpcId:testTagEpcId aTagEpcMask:tagMask aStatusMessage:&statusMessage];
```

Access Operations

The SDK supports performing of read, write, lock, kill, block erase, and block perma lock access operations on a specific tag. Access operations are performed via *srfidReadTag*, *srfidWriteTag*, *srfidLockTag*, *srfiKillTag*, *srfidBlockErase*, and *srfidBlockPermaLock* API functions accordingly. There are two versions of each API. One version targets the tag to access by using the Tag ID in the EPC Memory Bank, and the other targets the tag to access by filtering on criteria in other memory banks. The mentioned API functions are performed synchronously; the corresponding operation is started immediately and is stopped once tag data is reported by RFID reader or after a 5 second timeout.

There are also async versions of the access operation APIs. The following async versions can be used to perform access operations without the application getting blocked for completion of the same.

- `srfidReadTagAsync`
- `srfidWriteTagAsync`
- `srfidLockTagAsync`
- `srfidKillTagAsync`
- `srfidBlockPermaLockAsync`
- `srfidBlockEraseAsync`

The following example demonstrates performing of read and write access operations on one of the tags being inventoried.

```
/* identifier of one of active RFID readers is supposed to be stored in m_ReaderId variable */
/* allocate object for storing results of access operation */
srfidTagData *access_result = [[srfidTagData alloc] init];
/* id of tag to be read */
NSString *tag_id = @"36420124102N012610R98V91";
/* an object for storage of error response received from RFID reader */
NSString *error_response = nil;
/* request to read 8 words from EPC memory bank of tag specified by tag _id */
SRFID_RESULT result = [apiInstance srfidReadTag:m_ReaderId aTagID:tag_id
aAccessTagData:&access_result aMemoryBank:SRFID_MEMORYBANK_EPC aOffset:0 aLength:8
aPassword:0x00 aStatusMessage:&error_response];
if (SRFID_RESULT_SUCCESS == result)
     { NSLog(@"Request succeed\n");
/* check result code of access operation */
if (NO == [access_result getOperationSucceed]) {
     NSLog(@"Read operation has failed with error: %@\n", [access_result
getOperationStatus]);
     }
     else {
         NSLog(@"Memory bank data: %@", [access_result getMemoryBankData]);
     }
}
else if (SRFID_RESULT_RESPONSE_ERROR == result) {
     NSLog(@"Error response from RFID reader: %@\n", error_response);
}
else if (SRFID_RESULT_RESPONSE_TIMEOUT == result){
     NSLog(@"Timeout occurred\n");
}
else
{
     NSLog(@"Request failed\n");
}
[access_result release];
access_result = [[srfidTagData alloc] init];
error_response = nil;
/* data to be written */
NSString *data = @"N20122014R1010364989126V";
/* request to write a data to a EPC memory bank of tag specified by tag_id */
result = [apiInstance srfidWriteTag:m_ReaderId aTagID:tag_id aAccessTagData:&access_result
aMemoryBank:SRFID_MEMORYBANK_EPC aOffset:0 aData:data aPassword:0x00 aDoBlockWrite:NO
aStatusMessage:&error_response];
if (SRFID_RESULT_SUCCESS == result)
     { NSLog(@"Request succeed\n");
     /* check result code of access operation */
     if (NO == [access_result getOperationSucceed]) {
         NSLog(@"Write operation has failed with error: %@\n", [access_result
getOperationStatus]);
     }
}
else if (SRFID_RESULT_RESPONSE_ERROR == result) {
     NSLog(@"Error response from RFID reader: %@\n", error_response);
}
else if (SRFID_RESULT_RESPONSE_TIMEOUT == result) {
     NSLog(@"Timeout occurred\n");
}
else {
     NSLog(@"Request failed\n");
}
[access_result release];
```

Access Criteria

Access Criteria can be used with Access Operations to target tag(s) by filtering on data in the specified memory bank. An *srfidAccessCriteria* object can be created and used with the access function APIs described in the previous section. The *srfidAccessCriteria* parameter identifies the tag on which the access command needs to be carried out by the SDK. The following example demonstrates how to create a *srfidAccessCriteria* object and set the information for Tag Filter 1. Tag Filter 2 is not used in this example. The TID memory bank is set as the memory bank to access and the hard-coded tag serial number 01767C20 is used as the data pattern to filter.

```
// initialize access criteria
srfidAccessCriteria *accessCriteria = [[srfidAccessCriteria alloc] init];

// setup tag filter 1
srfidTagFilter *tagFilter1 = [[[srfidTagFilter alloc] init] autorelease];
[tagFilter1 setFilterMaskBank:SRFID_MEMORYBANK_TID];
[tagFilter1 setFilterData:@"01767C20"];
[tagFilter1 setFilterDoMatch:YES];
[tagFilter1 setFilterMask:@"FFFFFFFF"];
[tagFilter1 setFilterMaskStartPos:2];
[tagFilter1 setFilterMatchLength:2];

// set tag filter 1
[accessCriteria setTagFilter1:tagFilter1];
```

The *srfidAccessCriteria* created above can be provided as input to any of the available Access Operation APIs.

Timeout Value

The SDK provides the srfidSetAccessCommandOperationWaitTimeout API to set the access command operation wait timeout value (in milliseconds) for a particular RFID reader. If this command is not used to set the timeout value for the reader, a default value of 5000 milliseconds is used. The timeout value set by this API applies to all access functions (srfidReadTag, srfidWriteTag, srfidKillTag, srfidLockTag, srfidBlockErase, and srfidBlockPermaLock). The timeout value is not persistent between sessions.

The following example demonstrates how to use the API to set the access command operation wait timeout for a particular reader:

```
// Specify the ID of the reader.
int activeReaderId = 1;

// Set the access command operation wait timeout value to 3000 milliseconds.
[m_RfidSdkApi srfidSetAccessCommandOperationWaitTimeout:activeReaderId aTimeoutMs:3000];
```

srfidSetUniqueTagReportConfiguration

```
srfidUniqueTagsReport* setUniqueTag = [[srfidUniqueTagsReport alloc]init];
[setUniqueTag setUniqueTagsReportEnabled:true];

NSString *statusMessage = nil;

SRFID_RESULT result = [sdkInstance srfidSetUniqueTagReportConfiguration:readerId
aUtrConfiguration:setUniqueTag aStatusMessage:&statusMessage];

if (SRFID_RESULT_SUCCESS == result) {
NSLog(@"Set Unique Tag Reporting succeeded\n");
}
else {
NSLog(@"Request failed\n");
}
```

srfidGetUniqueTagReportConfiguration

```
srfidUniqueTagsReport* __autoreleasing getUniqueTag = [[srfidUniqueTagsReport alloc]init];

NSString *statusMessage = nil;
SRFID_RESULT result = [sdkInstance srfidGetUniqueTagReportConfiguration:readerId
aUtrConfiguration:&getUniqueTag aStatusMessage:&statusMessage];
```

srfidAuthenticate

```
NSString *status = [[NSString alloc]init];
SRFID_RESULT result = [sdkInstance srfidAuthenticate:readerId aAccessCriteria:nil
aAccessConfig:accessConfig aPassword:1 aMsgLength:96 aMsgData:@"0000965644023757796C69664"
aRespLength:128 aCsi:0 aDoSendRes:true aDoIncresplen:true aStatusMessage:&status];
```

srfidUntraceable

srfidUntraceableConfig *untraceConfig = [[srfidUntraceableConfig alloc]init];

NSString *status = [[NSString alloc]init];

[untraceConfig setShowEpc:true];

[untraceConfig setEpcLen:2];

[untraceConfig setShowUser:false];

SRFID_RESULT result = [sdkInstance srfidUntraceable:readerId aAccessCriteria:nil aAccessConfig:nil aPassword:01 aUntraceableConfig:untraceConfig aStatusMessage:&status];

srfidReadBuffer

```
NSString *status = [[NSString alloc]init];

SRFID_RESULT result = [sdkInstance srfidReadBuffer:readerId aAccessCriteria:nil
aAccessConfig:nil aPassword:1 aWordPtr:0 aBitCount:16 aStatusMessage:&status];
```

srfidSetCryptoSuite

```
NSString *status = [[NSString alloc]init];

SRFID_RESULT result = [sdkInstance srfidSetCryptoSuite:readerId aAccessCriteria:nil
aAccessConfig:nil aPassword:01 aKeyId:1 aIChallenge:@"96564402375796C69664"
aIncCustom:true aProfile:2 aOffset:2 aBlockCount:1 aProtMode:1 aStatusMessage:&status];
```

srfidStopOperation

```
NSString *statusMessage = [[NSString alloc]init];

SRFID_RESULT result = [sdkInstance srfidStopOperation:readerId
aStatusMessage:&statusMessage];
```

Chapter 4 GETTING STARTED WITH THE ZEBRA RFID DEMO APPLICATION and RFID API3 SDK for ANDROID

Introduction

This provides step-by-step instructions to import and run Zebra RFID demo application code and instructions to import the RFID API3 SDK module to build an application to work with the RFD8500.

Installing Android Studio

To install Android Studio go to http://developer.android.com/sdk/index.html and click DOWNLOAD ANDROID STUDIO 2.1.

Required packages for building source:

- The project uses the following configurations.
  - Minimum SDK Version - v19 (Android 4.4.2 KitKat)
  - Target SDK Version - v19 (Android 4.4.2 KitKat)
  - Gradle Version - v1 (2.2.1)
  - Java Version - Java7

Though the latest SDK tools packaged with Android Studio are acceptable, you can use the SDK manager to download the required Android SDK packages (Menu Tools > Android > SDK Manager).

Importing the Zebra RFID Demo Application Project

To import the Demo Application Project:

1. Open Android Studio. The following screen displays.

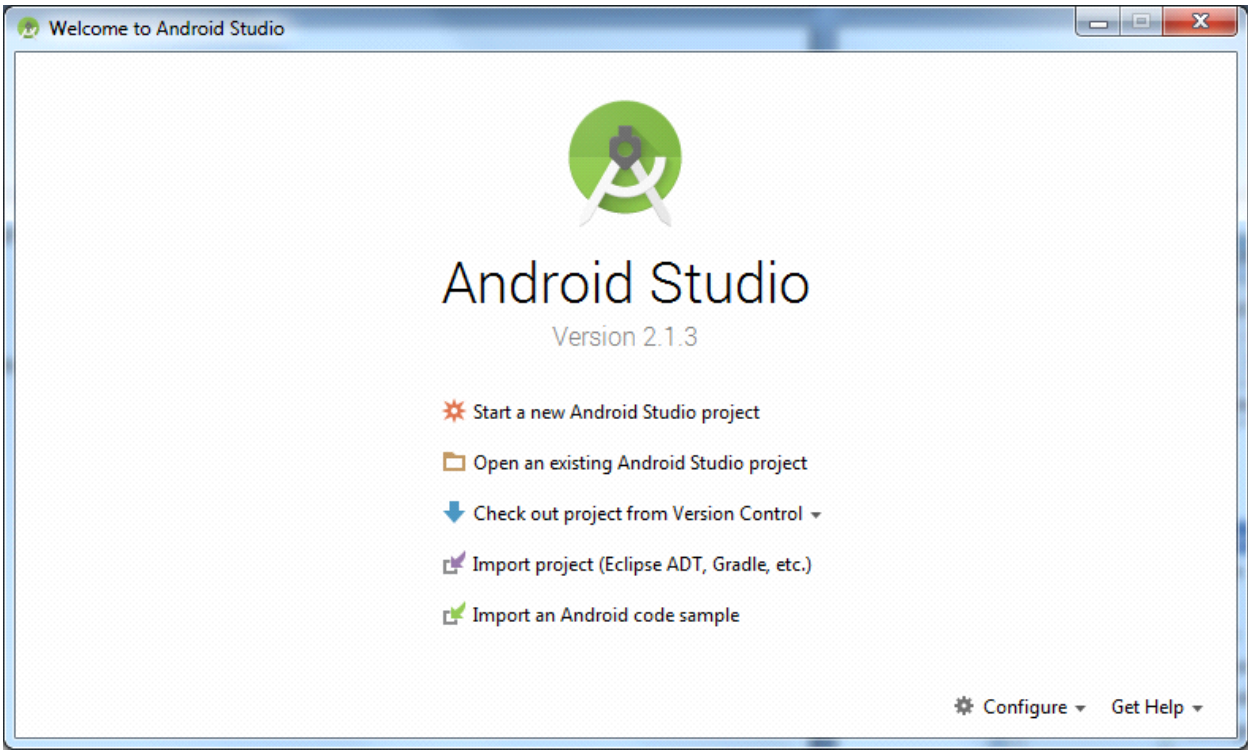

Figure 4-1 *Android Studio Main Screen*

2. Click **Import project** to set language and the SDK tool path.

3. Select **RFID Reader X** (Demo application folder).

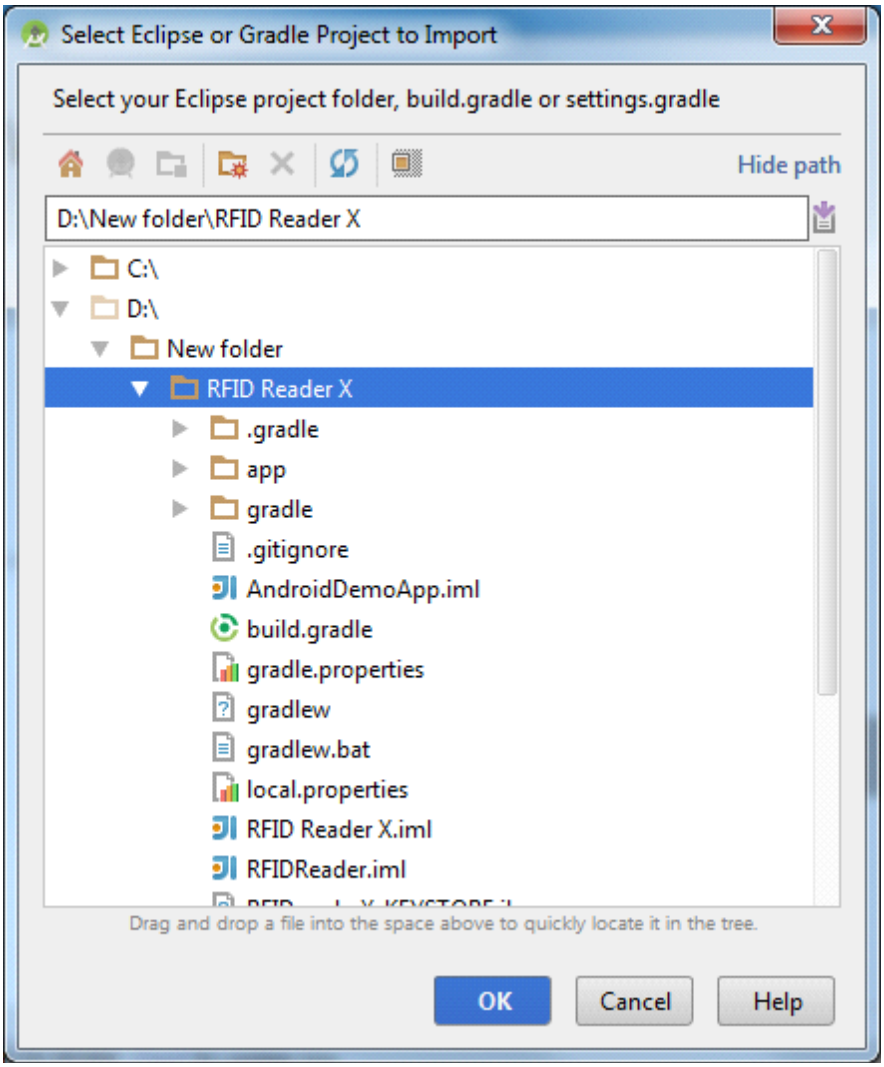

Figure 4-2 *Project Folder*

4. Android Studio automatically synchronizes the SDK path if required.

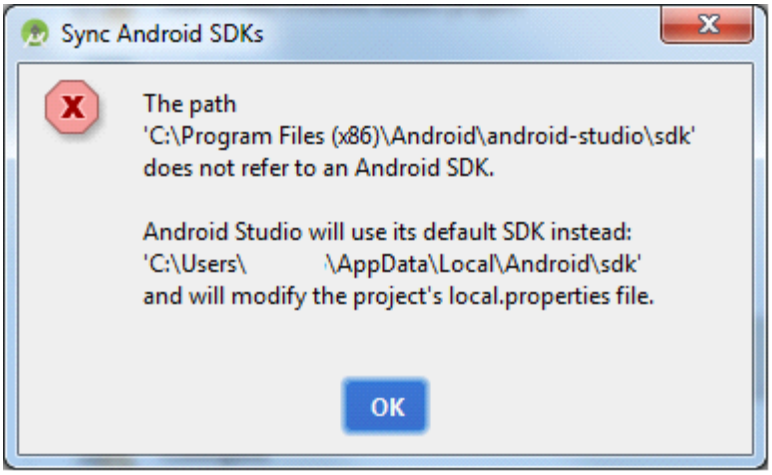

Figure 4-3 *Syncing Android SDKs*

Building and Running a Project

To build and run a project:

1. Android Studio may start downloading required Gradle/SDK packages first. Define proxy information in Android Studio and gradle.properties as required and then Android Studio starts building the project.

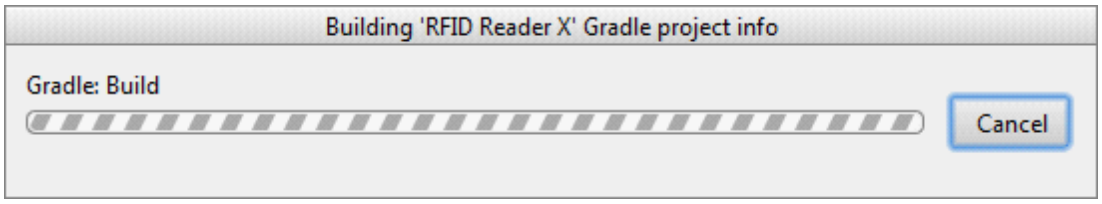

Figure 4-4 *Building the Project*

2. If prompted with a **Language Level Changed** dialog box, click **Yes**.

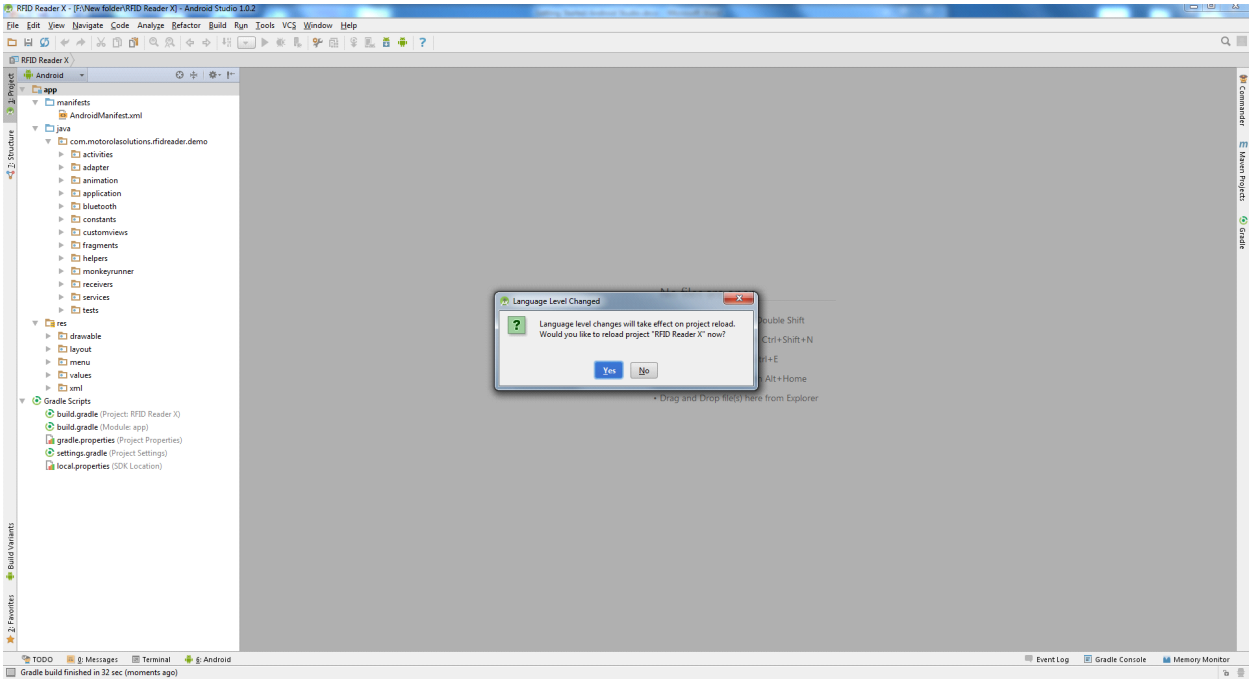

Figure 4-5 *Language Level Changed Dialog Box*

3. After completion of a successful compilation, run the application using the **run app** button. Android studio prompts for the deployment target. Install the built application on the target device by choosing a running device from the connected device list.

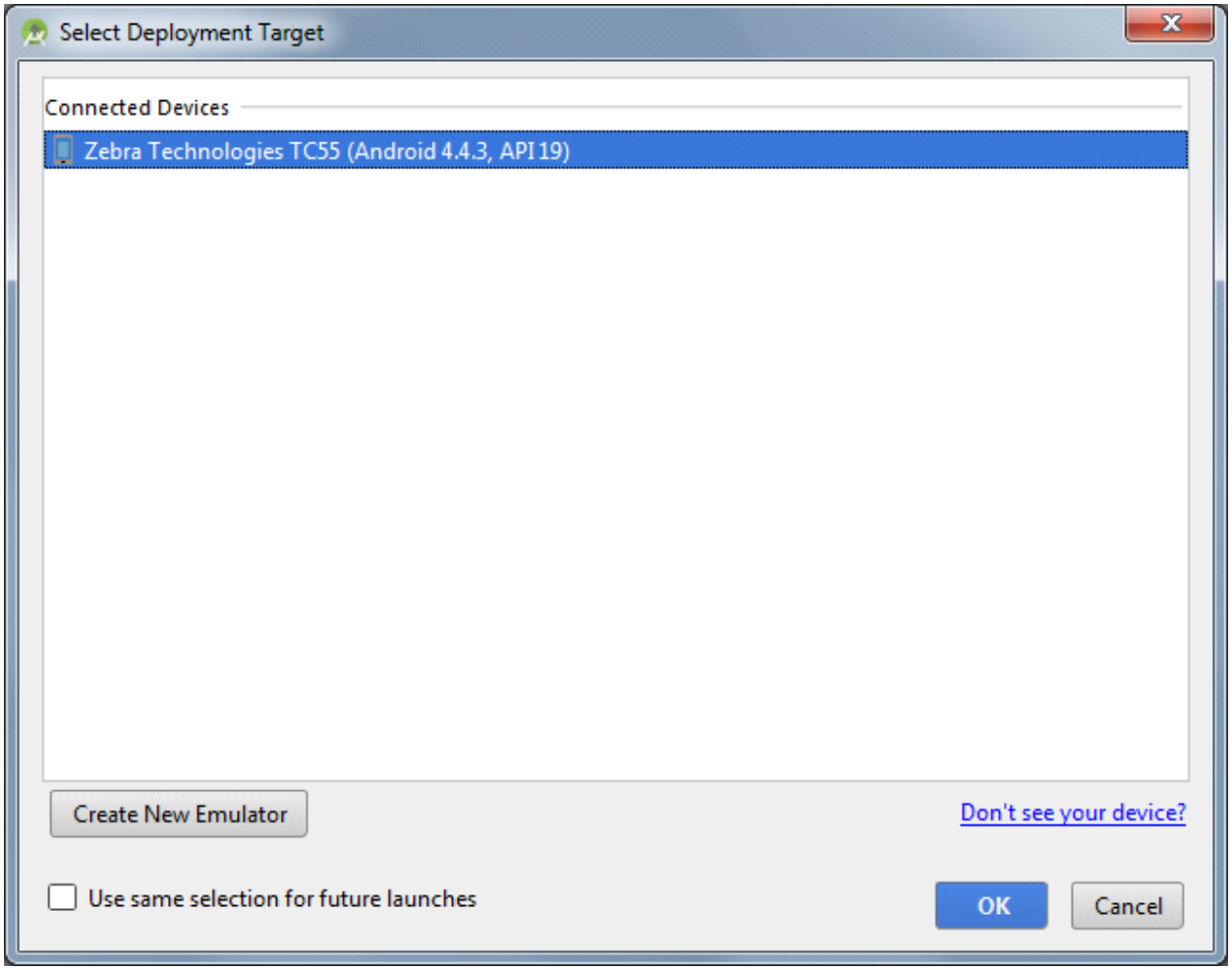

Figure 4-6 *Choose Device*

4. The screen in *Figure 4-7* shows the captured image of the demo application running on an Android device.

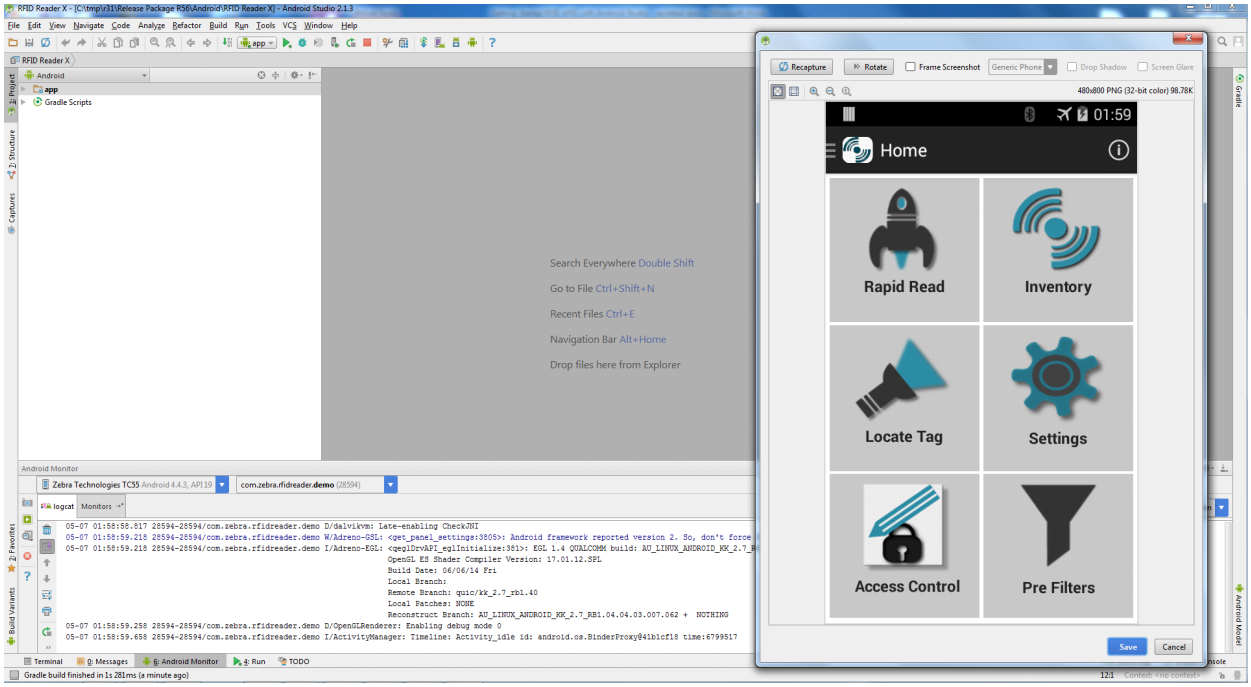

Figure 4-7 *Demo Application*

RFID API3 Android SDK

RFID API3 SDK for Android is provided in 'aar' package format. Copy the 'RFIDAPI3Library' folder to a local path.

Creating an Android Project

To create an Android Project:

1. Select File > New > New Project to create a new Android project and follow the on screen steps in the Android Studio New Project wizard.

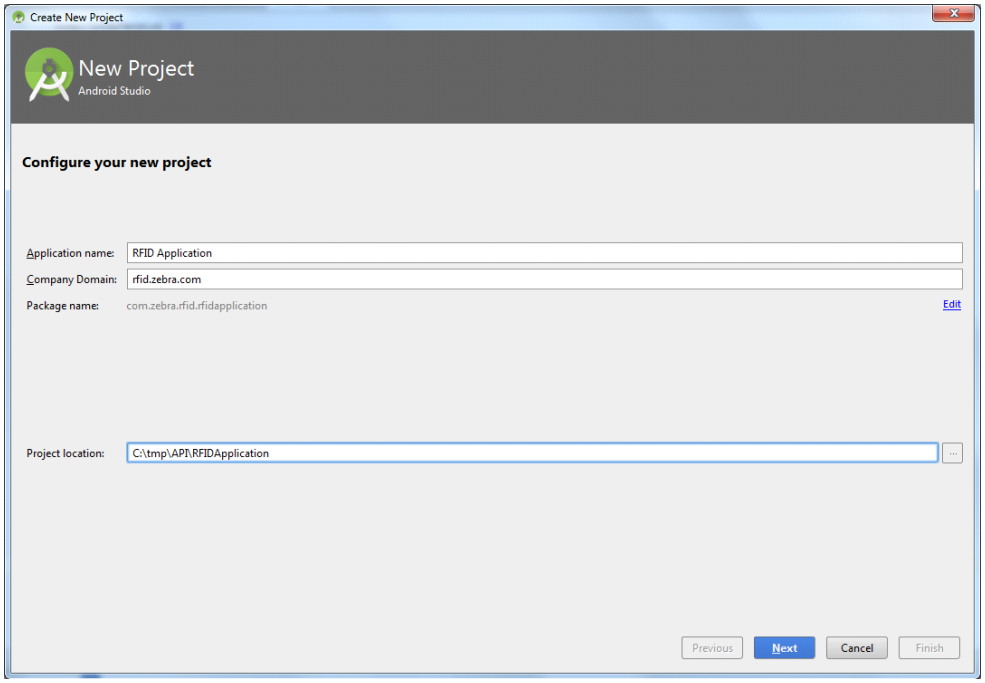

Figure 4-8 *Create New Project*

2. Navigate to File > New > Import Module to import the RFID API3 module.

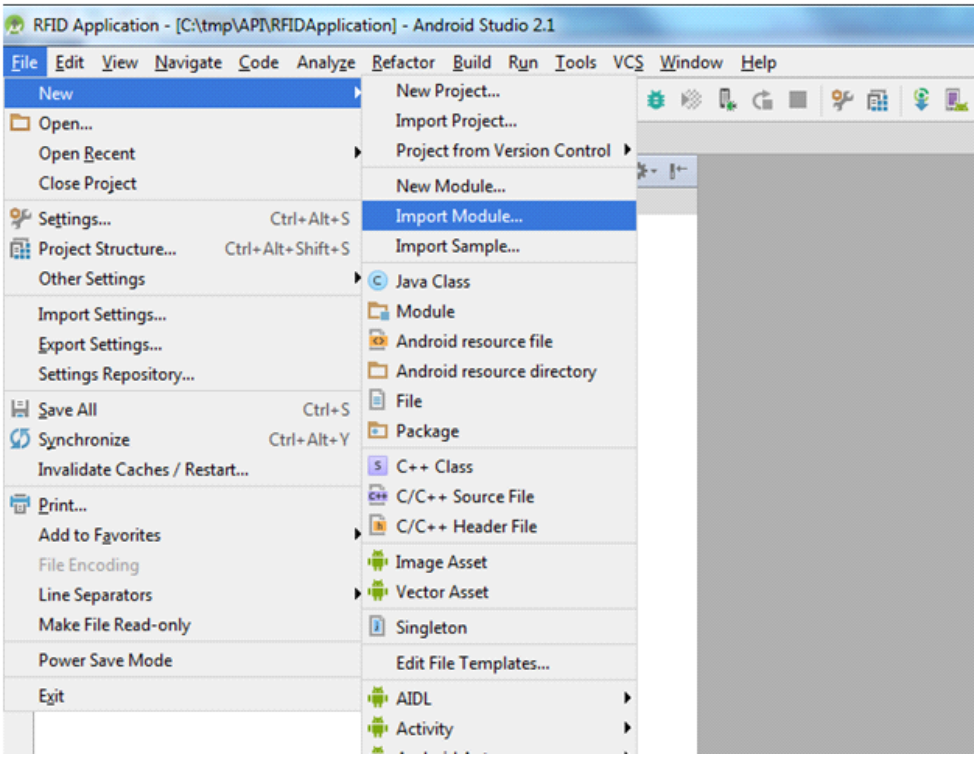

Figure 4-9 *Import RFID API3 Module*

3. Browse the RFIDAPI3Library folder. (After source directory selection the module name displays as RFIDAPI3Library.)

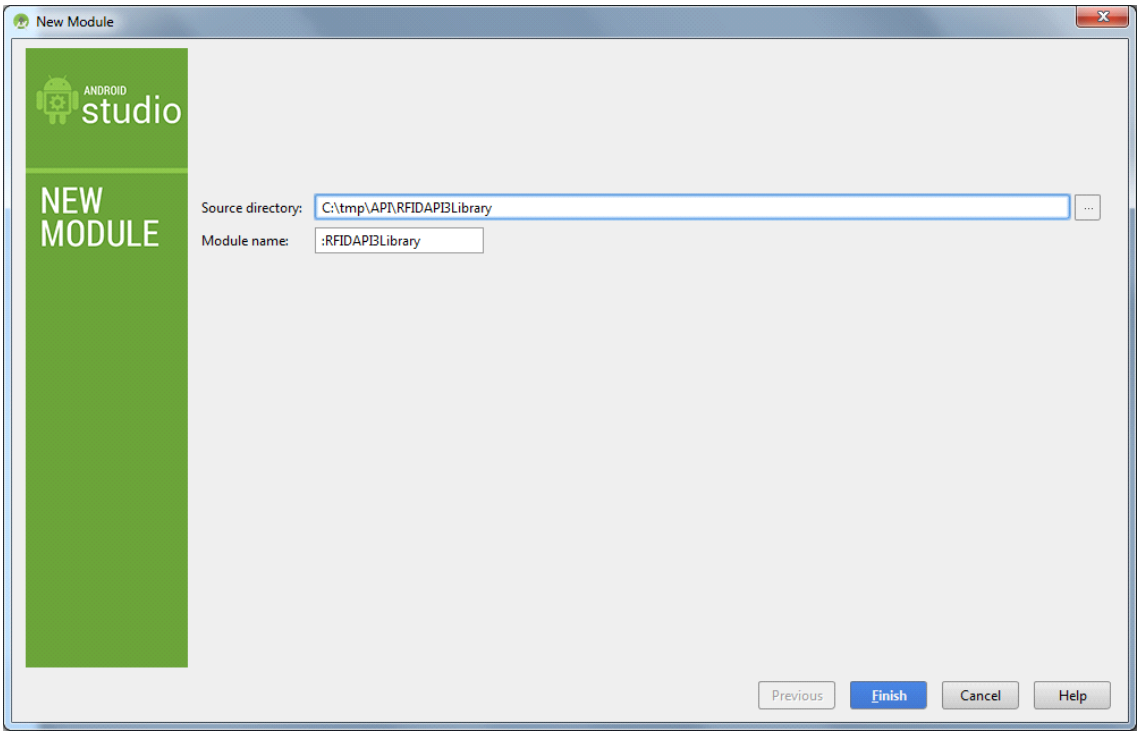

Figure 4-10 *New Module*

4. Add the module under dependencies in the application gradle file as shown in *Figure 4-11*.

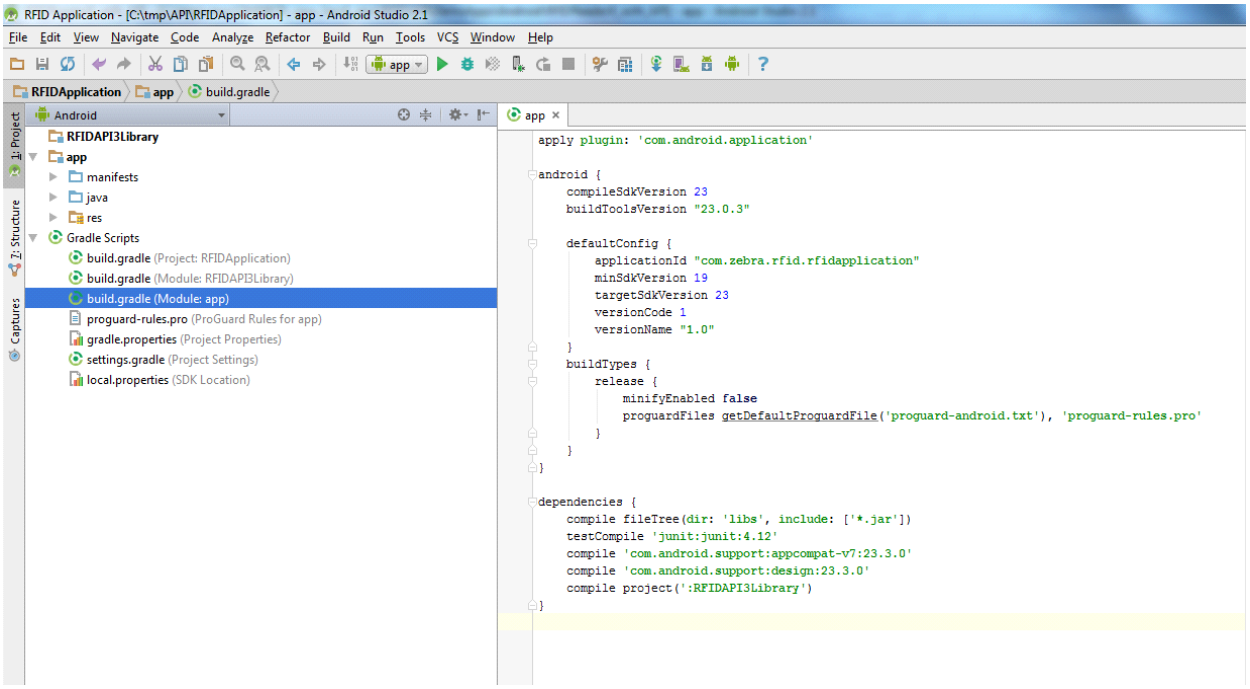

Figure 4-11 *Application Gradle Modification for Module*

5. Now the application is ready to import the com.zebra.rfid.api3.* package/class.

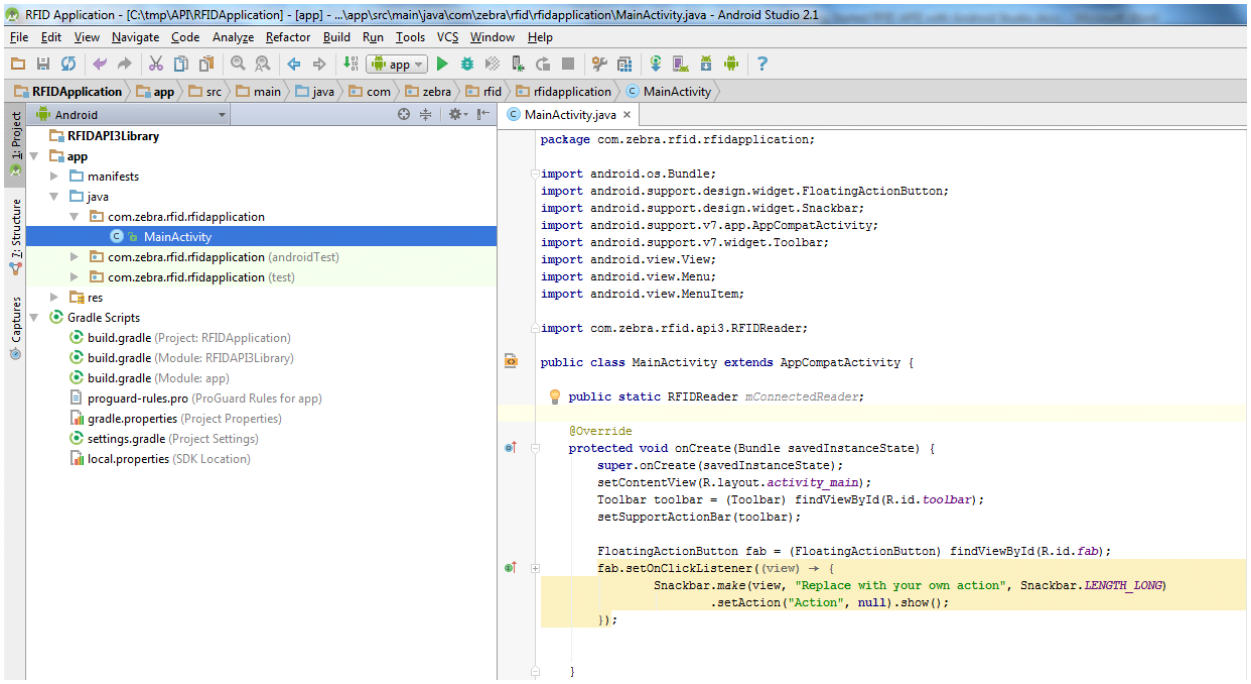

Figure 4-12 *Application MainAcitivity.java with Imported RFID API3 Package*

Chapter 5 ZEBRA RFID SDK for Android

Introduction

This chapter provides detailed information about how to use various functionality of SDK from basic to advance to develop Android application using the Zebra RFID SDK for Android.

The Zebra RFID SDK for Android allows applications to communicate with RFID readers that support the Zebra Easy Text interface and are connected to an Android device wirelessly via Bluetooth.

The Zebra RFID SDK for Android provides the API that can be used by external applications to manage connections of remote RFID readers, and to control connected RFID readers.

Basics

NOTE Detailed API documentation can be found in Java Class Reference Guide distributed with the Zebra RFID SDK for Android (see *Related Documents on page viii*.)

The Zebra RFID SDK for Android provides the ability to manage RFID readers' connections, perform various operations with connected RFID readers, configure connected RFID readers and knowing other information related to connected RFID readers.

Zebra RFID Android SDK consists of a static android library in 'aar' format that is supposed to be linked with an external Android application. Step-by-step instructions for configuring the Android application project to enable utilization of Zebra RFID Android SDK are provided in *Chapter 4, GETTING STARTED WITH THE ZEBRA RFID DEMO APPLICATION and RFID API3 SDK for ANDROID*.

All available APIs are defined under the com.zebra.rfid.api3 package. RFIDReader is the root Java class in the SDK. The application uses a single instance of an RFIDReader object to interact with a particular reader.

Use available readers and the RFIDReader object to register for events, connect with readers, and after successful connection, perform required operations such as inventory, access and locate.

It is recommended that all API calls should be made using Android 'AsyncTask' so that operations are performed in the background thread keeping the UI thread free.

If the API call fails, the API throws an exception. The application should call all APIs in try-catch block for handling exceptions.

Connection Management

Connect to an RFID Reader

Connection is the first step to talk to an RFID reader. Importing package is the first step to use any API. Import the package as shown below.

```
import com.mot.rfid.api3.*;
```

The Readers class instance gives a list of all available/paired RFID readers with an Android device. Readers list is in the form of ReaderDevice class.

```
ArrayList<ReaderDevice> availableRFIDReaderList =
readers.GetAvailableRFIDReaderList();
ReaderDevice readerDevice = availableRFIDReaderList.get(0);
```

ReaderDevice class includes instance of RFIDReader class; which is root class to interface and performing all operations with RFID reader.

```
RFIDReader rfidReader = readerDevice.getRFIDReader();
```

There are two ways to connect with the reader; either using RFIDReader class instance returned by above method or instantiating RFIDReader with Bluetooth name of the RFID reader.

```
// Establish connection to the RFID Reader
rfidReader.connect();
```

In addition, the application can implement Readers.RFIDReaderEventHandler in the following way to get notified for RFID reader getting BT paired (added) / unpaired (removal).

```
public class MainActivity extends ActionBarActivity implements
Readers.RFIDReaderEventHandler {
    @Override
    public void RFIDReaderAppeared(ReaderDevice device) {
        // handle reader addition
}
    @Override
    public void RFIDReaderDisappeared(ReaderDevice device) {
        // handle reader removal
    }
}
```

Special Connection Handling Cases

In a normal scenario, the reader connects fine, but the following are the cases which require special handling at the time of connection.

The following example shows connection is handled under try-catch block and operationfailure exception is thrown by connection API is stored and used for further analysis.

```
private OperationFailureException ex;
try {
// Establish connection to the RFID Reader
rfidReader.connect();
} catch (InvalidUsageException e) {
    e.printStackTrace();
} catch (OperationFailureException e) {
    e.printStackTrace();
    ex = e;
}
```

Region Is Not Configured

If the region is not configured then exception gives `RFID_READER_REGION_NOT_CONFIGURED` result;

Then caller gets supported regions and chooses operation regulatory region from list. Set region with required configurations.

```
if (ex.getResults() == RFIDResults.RFID_READER_REGION_NOT_CONFIGURED) {
     // Get and Set regulatory configuration settings

     // RegulatoryConfig regulatoryConfig = rfidReader.Config.getRegulatoryConfig();

     // RegionInfo regionInfo =
rfidReader.ReaderCapabilities.SupportedRegions.getRegionInfo(1);

     // regulatoryConfig.setRegion(regionInfo.getRegionCode());

     // rfidReader.Config.setRegulatoryConfig(regulatoryConfig);

}
```

Reader Requires BT Password

If the Bluetooth connection password is configured for connection, direct call to connection results exception pointing that BT password is required.

Set password in RFIDReader instance and then call connect again.

```
if (ex.getResults() == RFIDResults.RFID_CONNECTION_PASSWORD_ERROR) {
       // get password showPasswordDialog();
rfidReader.setPassword(password);
rfidReader.connect();
}
```

Reader Is Running In Batch Mode

If the reader is configured for auto or batch mode is enabled then reader keeps performing the inventory operation even it is not connected to Android device.

At that time, if connection is made to the particular RFID reader, then connection API throws the following exception, then application should stop the operation as required by usage

```
if (ex.getResults() == RFIDResults.RFID_BATCHMODE_IN_PROGRESS) {
     // handle batch mode related stuff
}
```

NOTE As the batch operation runs on the reader, reader related information cannot be retrieved by SDK autonomously.

It is the application responsibility to retrieve/update reader related information using the following API once the BATCH operation is stopped before any other use.

`rfidReader.PostConnectReaderUpdate();`

Disconnect

When the application is done with the connection and with the operations on the RFID reader, it calls following API to close the connection and to release and clean up the resources.

```
// Disconnects reader and performs clean-up
rfidReader.disconnect();
```

Reconnect

In case of abrupt disconnection from the reader side mainly because Bluetooth connectivity is challenged in out of range scenarios, the application can use the following API to reconnect with reader.

The application can try retries by periodically calling the reconnect API for finite number of times.

```
rfidReader.reconnect();
```

NOTE In case of reconnection with the reader, it is advisable to retrieve the configuration again, and registration of events, etc. It is also possible that the reader may have gone to batch mode if the reader was configured for same.

Knowing the Reader Capabilities

The capabilities (or Read-Only properties) of the reader can be known using the ReaderCapabilites class. The reader capabilities include the following:

General Capabilities

- Firmware Version - property.
- Model Name.
- Number of antennas supported.
- Tag Event Reporting Supported - Indicates the reader’s ability to report tag visibility state changes (New Tag, Tag Invisible, or Tag Visibility Changed).
- RSSI Filter Supported - Indicates the reader’s ability to report tags based on the signal strength of the back-scattered signal from the tag.
- NXP Commands Supported - Indicates whether the reader supports NXP commands like Change EAS,set Quiet, Reset Quiet,Calibrate.
- Tag LocationingSupported - Indicates the reader’s ability to locate a tag.
- DutyCycleValues - List of DutyCycle percentages supported by the reader.

Gen2 Capabilities

- Block Erase - supported
- Block Write - supported
- State Aware Singulation - supported
- Maximum Number of Operation in Access Sequence
- Maximum Pre-filters allowable per antenna
- RF Modes.

Regulatory Capabilities

- Country Code
- Communication Standard

UHF Band Capabilities

- Transmit Power table.
- Hopping enabled.
- Frequency Hop table (If hopping is enabled, this table has the frequency information.).
- Fixed Frequency table (If hopping is not enabled, this table contains the frequency list used by the reader. The one-based position of a frequency in this list is its channel index.).

Reader Identification

Reader ID and Reader ID Type (the reader identification can be MAC or EPC).

```
// Get Reader capabilities
System.out.println("\nReader ID: " + reader.ReaderCapabilities.ReaderID.getID());
System.out.println("\nModelName: " + reader.ReaderCapabilities.getModelName());
System.out.println("\nCommunication Standard: " +
reader.ReaderCapabilities.getCommunicationStandard().toString());
System.out.println("\nCountry Code: " + reader.ReaderCapabilities.getCountryCode());
System.out.println("\nFirwareVersion: " + reader.ReaderCapabilities.getFirwareVersion());
System.out.println("\nRSSI Filter: " + reader.ReaderCapabilities.isRSSIFilterSupported());
System.out.println("\nTag Event Reporting: " +
reader.ReaderCapabilities.isTagEventReportingSupported());
System.out.println("\nTag Locating Reporting: " +
reader.ReaderCapabilities.isTagLocationingSupported());
System.out.println("\nNXP Command Support: " +
reader.ReaderCapabilities.isNXPCommandSupported());
System.out.println("\nBlockEraseSupport: " +
reader.ReaderCapabilities.isBlockEraseSupported());
System.out.println("\nBlockWriteSupport: " +
reader.ReaderCapabilities.isBlockWriteSupported());
System.out.println("\nBlockPermalockSupport: " +
reader.ReaderCapabilities.isBlockPermalockSupported());
System.out.println("\nRecommisionSupport: " +
reader.ReaderCapabilities.isRecommisionSupported());
System.out.println("\nWriteWMISupport: " +
reader.ReaderCapabilities.isWriteUMISupported());
System.out.println("\nRadioPowerControlSupport: " +
reader.ReaderCapabilities.isRadioPowerControlSupported());
System.out.println("\nHoppingEnabled: " + reader.ReaderCapabilities.isHoppingEnabled());
System.out.println("\nStateAwareSingulationCapable: " +
reader.ReaderCapabilities.isTagInventoryStateAwareSingulationSupported());
System.out.println("\nUTCClockCapable: " +
reader.ReaderCapabilities.isUTCClockSupported());
System.out.println("\nNumOperationsInAccessSequence: " +
reader.ReaderCapabilities.getMaxNumOperationsInAccessSequence());
System.out.println("\nNumPreFilters: " + reader.ReaderCapabilities.getMaxNumPreFilters());
System.out.println("\nNumAntennaSupported: " +
reader.ReaderCapabilities.getNumAntennaSupported());
System.out.println("\nNumGPIPorts: " + reader.ReaderCapabilities.getNumGPIPorts());
System.out.println("\nNumGPIPorts: " + reader.ReaderCapabilities.getNumGPOPorts());
```

Configuring the Reader

RF Mode

The reader has one or more sets of C1G2 RF Mode that the reader is capable of operating. The supported RF mode can be retrieved from the RF Mode table using the ReaderCapabilities class.

The API `getRFModeTableInfo` in reader capabilities.RFModes gets the RF Mode table from the reader. The `getLinkedProfiles` function described below populates the RFModeTable into an ArrayList.

```
// The rfModeTable is populated by the getRFModeTableInfo function

  public RFModeTable rfModeTable =
reader.ReaderCapabilities.RFModes.getRFModeTableInfo(0);

  // The linked profiles are added to an Arraylist

  private void getLinkedProfiles(ArrayList<String> linkedProfiles) {

  RFModeTableEntry rfModeTableEntry = null;

  for (int i = 0; i < rfModeTable.length(); i++) {

  rfModeTableEntry = rfModeTable.getRFModeTableEntryInfo(i);

  linkedProfiles.add(rfModeTableEntry.getBdrValue() + " " +
rfModeTableEntry.getModulation() + " " + rfModeTableEntry.getPieValue() + " " +

  fModeTableEntry.getMaxTariValue() + " " + rfModeTableEntry.getMaxTariValue() + " " +
rfModeTableEntry.getStepTariValue());

    }
 }
```

Antenna Specific Configuration

The config class contains the Antennas object. The individual antenna can be accessed and configured using the index.

Antenna Configuration

The AntennaProperties is used to set the antenna configuration to individual antenna or all the antennas.

The antenna configuration (`SetAntennaConfig` function) is comprised of Antenna ID, Receive Sensitivity Index, Transmit Power Index, and Transmit Frequency Index. These indexes are refers to the Receive Sensitivity table, Transmit Power table, Frequency Hop table, or Fixed Frequency table, respectively. These tables are available in Reader capabilities. `getAntennaConfig` function gets the antenna configuration for the given antenna ID.

RF Configuration

The function `setAntennaRfConfig` is added to configure antenna RF configuration to individual antenna. This function is similar to `setAntennaConfig` but includes additional parameters specific pertaining to antenna.

The configuration includes Receive Sensitivity Index, Transmit Power Index, Transmit Frequency Index, and RF Mode Table Index. These indexes are refers to the Receive Sensitivity table, Transmit Power table, Frequency Hop table, Fixed Frequency table, or RF Mode table, respectively. These tables are available in Reader capabilities. Also, includes tari, transmit port, receive port and Antenna Stop trigger condition. The stop condition can be 'n' number of attempts, duration based. The function `getAntennaRfConfig` gets the antenna RF configuration for the given antenna ID.

```
Antennas.AntennaRfConfig antennaRfConfig = reader.Config.Antennas.getAntennaRfConfig(1);

antennaRfConfig.setrfModeTableIndex(4);

antennaRfConfig.setTari(270);

antennaRfConfig.setTransmitPowerIndex(7);

reader.Config.Antennas.setAntennaRfConfig(1,antennaRfConfig);
```

Singulation Control

The function `getSingulationControl` retrieves the current settings of the singulation control from the reader for the given Antenna ID.

To set the singulation control settings, the `setSingulationControl` method is used. The following settings can be configured:

- Session: Session number to use for inventory operation.
- Tag Population: An estimate of the tag population in view of the RF field of the antenna.
- Tag Transit Time: An estimate of the time a tag typically remains in the RF field.
- State Aware Singulation Action: The action includes the Inventory state and SL flag. The action can be used if only the reader supports this capability. The ReaderCapabilities class helps to determine whether state-aware singulation is supported or not.

```
// Get Singulation Control for the antenna 1

Antennas.SingulationControl singulationControl;

singulationControl = reader.Config.Antennas.getSingulationControl(1);

// Set Singulation Control for the antenna 1

Antennas.SingulationControl singulationControl;

singulationControl.setSession(SESSION.SESSION_S0);

singulationControl.setTagPopulation((short) 30);

singulationControl.Action.setSLFlag(SL_FLAG.SL_ALL);

singulationControl.Action.setInventoryState(INVENTORY_STATE.INVENTORY_STATE_A);

reader.Config.Antennas.setSingulationControl(1, singulationControl);
```

Tag Report Configuration

The SDK provides an ability to configure a set of fields to be reported in a response to an operation by a specific active RFID reader.

Supported fields that might be reported include the following:

- First seen time
- Last seen time
- PC value
- RSSI value
- Phase value
- Channel index
- Tag seen count.

The function `getTagStorageSettings` retrieves the tag report parameters from the reader for the given Antenna ID.

To set the Tag report parameters, the `setTagStorageSettings` method is used. The following settings can be configured:

```
// Get tag storage settings from the reader

TagStorageSettings tagStorageSettings = reader.Config.getTagStorageSettings();
// set tag storage settings on the reader with all fields

tagStorageSettings.setTagFields(TAG_FIELD.ALL_TAG_FIELDS);

reader.Config.setTagStorageSettings(tagStorageSettings);
```

Regulatory Configuration

The SDK supports managing of regulatory related parameters of a specific active RFID reader.

Regulatory configuration includes the following:

- Code of selected region.
- Hopping.
- Set of enabled channels.

A set of enabled channels includes only such channels that are supported in the selected region. If hopping configuration is not allowed for the selected regions, a set of enabled channels is not specified.

Regulatory parameters could be retrieved and set via `getRegulatoryConfig` and `setRegulatoryConfig` API functions accordingly. The region information is retrieved using `getRegionInfo` API. The following example demonstrates retrieving of current regulatory settings and configuring the RFID reader to operate in one of supported regions.

```
// Get and Set regulatory configuration settings

RegulatoryConfig regulatoryConfig = reader.Config.getRegulatoryConfig();

RegionInfo regionInfo = reader.ReaderCapabilities.SupportedRegions.getRegionInfo(1);

regulatoryConfig.setRegion(regionInfo.getRegionCode());

regulatoryConfig.setIsHoppingOn(regionInfo.isHoppingConfigurable());

regulatoryConfig.setEnabledChannels(regionInfo.getSupportedChannels());

reader.Config.setRegulatoryConfig(regulatoryConfig);
```

Beeper Configuration

The SDK provides an ability to configure a beeper of a specific active RFID reader. The beeper could be configured to one of predefined volumes (low, medium, high) or be disabled. Retrieving and setting of beeper configuration is performed via `getBeeperVolume` and `setBeeperVolume` API functions as demonstrated in the following example.

```
// get Beeper volume settings

BEEPER_VOLUME beeperVolume = reader.Config.getBeeperVolume();

// Set Beeper volume settings

reader.Config.setBeeperVolume(BEEPER_VOLUME.HIGH_BEEP);
```

Dynamic Power Management Configuration

The SDK provides a way to configure the reader to operate in dynamic power mode. The dynamic power state can be switched to be either on or off.

```
// set Dynamic power state on

reader.Config.setDPOState(DYNAMIC_POWER_OPTIMIZATION.ENABLE);

// set Dynamic power state off

reader.Config.setDPOState(DYNAMIC_POWER_OPTIMIZATION.DISABLE);
```

Saving Configuration

Various parameters of a specific RFID reader configured via SDK are lost after the next power down. The SDK provides an ability to save a persistent configuration of RFID reader. The `saveConfig API` function can be used to make the current configuration persistent over power down and power up cycles. The following example demonstrates utilization of mentioned API functions.

```
// Saving the configuration

reader.Config.saveConfig();
```

Managing Events

The Application can register for one or more events so as to be notified of the same when it occurs. There are basically two main events.

- `eventReadNotify`: Notifies whenever read tag event occurs with read tag data as argument. By default, the event comes with tag data. If not required, we can disable using function setAttachTagDataWithReadEvent.
- `eventStatusNotify`: Notifies whenever status event occurs with status event data as argument.

Table 5-1 *Events*

Event	Description
GPI_EVENT	Not supported in Android RFID SDK.
BUFFER_FULL_WARNING_EVENT	When the internal buffers are 90% full, this event is signaled.
ANTENNA_EVENT	Not supported in Android RFID SDK.
INVENTORY_START_EVENT	Inventory Operation started. In case of periodic trigger, this event is triggered for each period.
INVENTORY_STOP_EVENT	Inventory Operation has stopped. In case of periodic trigger this event is triggered for each period.
ACCESS_START_EVENT	Not supported in Android RFID SDK.
ACCESS_STOP_EVENT	Not supported in Android RFID SDK.
DISCONNECTION_EVENT	Event notifying disconnection from the Reader. The Application can call reconnect method periodically to attempt reconnection or call disconnect method to cleanup and exit.
BUFFER_FULL_EVENT	When the internal buffers are 100% full, this event is signaled and tags are discarded in FIFO manner.
NXP_EAS_ALARM_EVENT	Not Supported in Android RFID SDK.
READER_EXCEPTION_EVENT	Event notifying that an exception has occurred in the Reader. When this event is signaled, StatusEventData.ReaderExceptionEventData.ReaderExceptionEventType can be called to know the reason for the exception which is coming as part of Events.StatusEventArgs. The Application can continue to use the connection if the reader renders is usable.
HANDHELD_TRIGGER_EVENT	A hand-held Gun/Button event Pull/Release has occurred.
TEMPERATURE_ALARM_EVENT	When Temperature reaches Threshold level, this event is generated. The event data contains source name (PA/Ambient), current Temperature and alarm Level (Low, High or Critical)
BATTERY_EVENT	Events notifying different levels of battery, state of the battery, if charging or discharging.
OPERATION_END_SUMMARY_EVENT	Event generated when operation end summary has been generated. The data associated with the event contains total rounds, total number of tags and total time in micro secs.
BATCH_MODE_EVENT	The batch mode event notification when the reader indicates whether batch mode is in progress.
POWER_EVENT	Events which notify the different power states of the reader device. The event data contains cause, voltage, current and power.

(continued on next page)

```
// registering for read tag data notification

EventHandler eventHandler = new EventHandler();
reader.Events.addEventsListener(eventHandler);

// Subscribe required status notification
myReader.Events.setInventoryStartEvent(true);
myReader.Events.setInventoryStopEvent(true);
// enables tag read notification. if this is set to false, no tag read notification is send
myReader.Events.setTagReadEvent(true);
myReader.Events.setReaderDisconnectEvent(true);
myReader.Events.setBatteryEvent(true);
myReader.Events.setBatchModeEvent(true);

// Read/Status Notify handler

   // Implement the RfidEventsLister class to receive event notifications
    class EventHandler implements RfidEventsListener {
        // Read Event Notification
        public void eventReadNotify(RfidReadEvents e){
            // Recommended to use new method getReadTagsEx for better performance in case
of large tag population
            TagData[] myTags = myReader.Actions.getReadTags(100);
            if (myTags != null)
            {
                for (int index = 0; index < myTags.length; index++)
                {
                    System.out.println("Tag ID " + myTags[index].getTagID());

                    if (myTags[index].getOpCode() ==
ACCESS_OPERATION_CODE.ACCESS_OPERATION_READ &&
                            myTags[index].getOpStatus() ==
ACCESS_OPERATION_STATUS.ACCESS_SUCCESS)
                    {
                        if (myTags[index].getMemoryBankData().length() > 0) {
                            System.out.println(" Mem Bank Data " +
myTags[index].getMemoryBankData());
                        }
                    }
                }
            }
        }
        // Status Event Notification
        public void eventStatusNotify(RfidStatusEvents e) {
            System.out.println("Status Notification: " +
e.StatusEventData.getStatusEventType());
        }
    }
// Unregistering for read tag data notification

reader.Events.removeEventsListener(eventHandler);
```

Device Status Related Events

Device status, like battery, power, and temperature, is obtained through events after initiating the following API

```
reader.Config.getDeviceStatus(battery, power, temperature)
```

Response to the above API comes as battery event, power event, and temperature event according to the set boolean value in the respective parameters.

The following is an example of how to get these events.

```
try {
    if (reader != null)
        reader.Config.getDeviceStatus(true, false, false);
    else
        stopTimer();
} catch (InvalidUsageException e) {
    e.printStackTrace();
} catch (OperationFailureException e) {
    e.printStackTrace();
}
```

Basic Operations

Tag Storage Settings

This section covers the basic/simple operations that an application would need to be performed on an RFID reader which includes inventory and single tag access operations.

The application needs to get the tags from the dll which are reported by the reader. Tags can be reported as part of an Inventory operation (reader.Actions.Inventory.perform) or a Read Access operation (reader.Actions.TagAccess.readEvent or reader.Actions.TagAccess.readWait).

Applications can also configure to receive tag reports that indicate the results of access operations as shown below.

```
TagStorageSettings tagStorageSettings = reader.Config.getTagStorageSettings();
tagStorageSettings.enableAccessReports ( true);
reader.Config.setTagStorageSettings(tagStorageSettings);
```

Each tag has a set of associated information along with it. During the Inventory operation the reader reports the EPC-ID of the tag where as during the Read-Access operation the requested Memory Bank Data is also reported apart from EPC-ID. In either case, there is additional information like PC-bits, RSSI, last time seen, tag seen count, etc. that is available for each tag. This information is reported to the application as TagData for each tag reported by the reader.

Applications can also choose to enable/disable reporting certain fields in TAG_DATA. Disabling certain fields can sometimes improve the performance as the reader and the dll are not processing that information. It can also result in specific behavior. For example, disabling reporting an Antenna Id can result in the application receiving a single unique tag even though they were multiple entries of the same tag reported from different antennas. The following snippet shows enabling the reporting of PeakRSSI, Tag Seen Count, PC and CRC only and disabling other fields like Antenna ID, Time Stamps, and XPC.

(continued on next page)

```
TagStorageSettings tagStorageSettings = reader.Config.getTagStorageSettings();
TAG_FIELD[] tagField = new TAG_FIELD[4];
tagField[0] = TAG_FIELD.PC;
tagField[1] = TAG_FIELD.PEAK_RSSI;
tagField[2] = TAG_FIELD.TAG_SEEN_COUNT;
tagField[3] = TAG_FIELD.CRC;
tagStorageSettings.setTagFields(tagField);
reader.Config.setTagStorageSettings(tagStorageSettings);
```

Following are a few use-cases that get tags from the reader.

Simple Inventory (Continuous)

A Simple Continuous Inventory operation reads all tags in the field of view of all antennas of the connected RFID reader. It uses no filters (pre-filters or post-filters) and the start and stop trigger for the inventory is the default (i.e., start immediately when reader.Actions.Inventory.perform is called, and stop immediately when reader.Actions.Inventory.stop is called).

```
// perform simple inventory

reader.Actions.Inventory.perform();

// Keep getting tags in the eventReadNotify event if registered

// stop the inventory

reader.Actions.Inventory.stop();
```

Simple Access Operations - On Single Tag

Tag Access operations can be performed on a specific tag or can be applied on tags that match a specific Access-Filter. If no Access-Filter is specified the Access Operation is performed on all tags in the field of view of chosen antennas.

This section covers the Simple Tag Access operation on a specific tag which could be in the field of view of any of the antennas of the connected RFID reader.

```
// dpo should be disabled before any access operation

reader.Config.setDPOState(DYNAMIC_POWER_OPTIMIZATION.DISABLE);
```

Read

The application can call method reader.Actions.TagAccess.readWait to read data from a specific memory bank.

```
// Read user memory bank for the given tag ID

String tagId = "1234ABCD00000000000025B1";
TagAccess tagAccess = new TagAccess();
TagAccess.ReadAccessParams readAccessParams = tagAccess.new ReadAccessParams();
TagData readAccessTag;
readAccessParams.setAccessPassword(0);
readAccessParams.setByteCount(8);
readAccessParams.setMemoryBank(MEMORY_BANK.MEMORY_BANK_USER);
readAccessParams.setByteOffset(0);
readAccessTag = reader.Actions.TagAccess.readWait(tagId, readAccessParams, null);
System.out.println(readAccessTag.getMemoryBank().toString() + " : " +
readAccessTag.getMemoryBankData());
```

Write, Block-Write

The application can call method reader.Actions.TagAccess.writeWait or reader.Actions.TagAccess.blockWriteWait to write data to a specific memory bank. The response is returned as a Tagdata from where number of words can be retrieved.

```
// Write user memory bank data

String tagId = "1234ABCD00000000000025B1";

TagAccess tagAccess = new TagAccess();

TagAccess.WriteAccessParams writeAccessParams = tagAccess.new WriteAccessParams();

byte[] writeData = new byte[] {0x11, 0x22, 0x33, 0x44};

writeAccessParams.setAccessPassword(0);

writeAccessParams.setWriteDataLength(writeData.Length);

writeAccessParams.setMemoryBank(MEMORY_BANK.MEMORY_BANK_USER);

writeAccessParams.setByteOffset(0);

writeAccessParams.setWriteData(writeData);

// antenna Info is null – performs on all antenna

TagData tagData = reader.Actions.TagAccess.writeWait(tagId, writeAccessParams, null);
```

Lock

The application can call method reader.Actions.TagAccess.lockWait to perform a lock operation on one or more memory banks with specific privileges.

```
// Lock the tag

String tagId = "1234ABCD00000000000025B1";

TagAccess tagAccess = new TagAccess();

TagAccess.LockAccessParams lockAccessParams = tagAccess.new LockAccessParams();

/* lock now */

lockAccessParams.setLockPrivilege(LOCK_DATA_FIELD.LOCK_USER_MEMORY,
LOCK_PRIVILEGE.LOCK_PRIVILEGE_READ_WRITE);

lockAccessParams.setAccessPassword(0);

reader.Actions.TagAccess.lockWait(tagId, lockAccessParams, null);
```

Kill

The application can call method reader.Actions.TagAccess.killWait to kill a tag.

```
// Kill the tag

String tagId = "1234ABCD00000000000025B1";

TagAccess tagAccess = new TagAccess();

TagAccess.KillAccessParams killAccessParams = tagAccess.new KillAccessParams();

killAccessParams.setKillPassword(0);

reader.Actions.TagAccess.killWait(tagId, killAccessParams, null);
```

Block-Erase

The application can call RFID_BlockErase to erase the contents of a tag.

```
// Block Erase

String tagId = "1234ABCD00000000000025B1";

TagAccess tagAccess = new TagAccess();

TagAccess.BlockEraseAccessParams blockEraseAccessParams = tagAccess.new
BlockEraseAccessParams();

blockEraseAccessParams.setAccessPassword(0);

blockEraseAccessParams.setMemoryBank(MEMORY_BANK.MEMORY_BANK_USER); // user memory bank

blockEraseAccessParams.setByteOffset(0);  // start erasing from offset 0

blockEraseAccessParams.setByteCount(16); // number of bytes to erase

reader.Actions.TagAccess.blockEraseWait(tagId, blockEraseAccessParams, null);
```

Block-Permalock

The application can call method Reader.Actions.TagAccess.blockPermalockWait to block a permalock tag. Tags reported as part of Block-Permalock access operation have TagData.getOpCode as ACCESS_OPERATION_BLOCK_PERMALOCK and TagData.getOpStatus indicating the result of the operation; if TagData.OpStatus is ACCESS_SUCCESS, TagData.getMemoryBankData contains the Block-Permalock Mask Data.

```
// Block-Perma Lock the tag

String tagId = "1234ABCD00000000000025B1";

TagAccess tagAccess = new TagAccess();

TagAccess.BlockPermalockAccessParams blockPermalockAccessParams = tagAccess.new
BlockPermalockAccessParams();

byte[] permalockMask = new byte[] {(byte)0xF0, 0x00};
```

(continued on next page)

```
blockPermalockAccessParams.setReadLock(true);

blockPermalockAccessParams.setMemoryBank(MEMORY_BANK.MEMORY_BANK_USER);

blockPermalockAccessParams.setByteOffset(0);

blockPermalockAccessParams.setByteCount(1);

blockPermalockAccessParams.setMask(permalockMask);

blockPermalockAccessParams.setMaskLength(2);

reader.Actions.TagAccess.blockPermalockWait(tagId, blockPermalockAccessParams, null);
```

Access Operations on Specific Memory Field of Single Tag

The following functions assist in writing to specific memory fields of a specific tag:

1. `writeTagIDWait` - This method writes to TagID of tag(s) and adjusts the PC bits according to the length of the TagID. When the TagID is modified, this API ensures that the tag subsequently backscatters the modified EPC, for which it also writes the length of the new or updated (PC + EPC) into the first 5 bits of the tag's PC.
2. `writeKillPasswordWait` - This method writes the kill password of the tag(s).
3. `writeAccessPasswordWait` - This method writes the access password of the tag(s).

Advanced Operations

Using Pre-Filters

Pre-filters are the same as the Select command of C1G2 specification. Once applied, pre-filters are applied prior to Inventory and Access operations.

Introduction

Singulation

Singulation refers to the method of identifying an individual Tag in a multiple-Tag environment. RFID readers could support State-Aware or State-Unaware pre-filtering (or singulation) which is indicated by the boolean flag `IsTagInventoryStateAwareSingulationSupported` in the ReaderCapabilities class.

In order to filter tags that match a specific condition, it is necessary to use the tag-sessions and their states (setting the tags to different states based on match criteria - `reader.Actions.PreFilters.add`) so that while performing inventory, tags can be instructed to participate (singulation - `reader.Config.Antennas.setSingulationControl`) or not participate in the inventory based on their states.

Sessions and Inventoried Flags

Tags provide four sessions (denoted S0, S1, S2, and S3) and maintain an independent inventoried flag for each session. Each of the four inventoried flags has two values, denoted A and B. These inventoried flag of each session can be set to A or B based on match criteria using method `reader.Actions.PreFilters.add`.

Selected Flag

Tags provide a selected flag, SL, which can be asserted or deasserted based on match criteria using method `reader.Actions.PreFilters.add`.

State-Aware Singulation

In state-aware singulation the application can specify detailed controls for singulation: Action and Target.

Action indicates whether matching Tags assert or deassert SL (Selected Flag), or set their inventoried flag to A or to B. Tags conforming to the match criteria specified using the method `reader.Actions.PreFilters.Add` are considered matching and the remaining are non-matching.

Target indicates whether to modify a tag's SL flag or its inventoried flag, and in the case of inventoried it further specifies one of four sessions.

Applying Pre-Filters

The following are the steps to use pre-filters:

- Add pre-filters
- Set appropriate singulation controls
- Perform Inventory or Access operation

Add Pre-filters

Each RFID reader supports a maximum number of pre-filters per antenna as indicated by `reader.ReaderCapabilites.getMaxNumPreFilters` which can be known using the ReaderCapabilities.

The application can set pre-filters using `reader.Actions.PreFilters.add` and remove using `reader.Actions.PreFilters.delete`.

State-Aware Settings

```
// Add state aware pre-filter

PreFilters filters = new PreFilters();

PreFilters.PreFilter filter = filters.new PreFilter();

byte[] tagMask = new byte[] { 0x12, 0x11 };

filter.setAntennaID((short)3);// Set this filter for Antenna ID 3

filter.setTagPattern(tagMask);// Tags which starts with 0x1211

filter.setTagPatternBitCount(tagMask.length * 8);

filter.setBitOffset(32); // skip PC bits (always it should be in bit length)

filter.setMemoryBank(MEMORY_BANK.MEMORY_BANK_EPC);
```

(continued on next page)

```
filter.setFilterAction(FILTER_ACTION.FILTER_ACTION_STATE_AWARE); // use state aware
singulation

filter.StateAwareAction.setTarget(TARGET.TARGET_INVENTORIED_STATE_S1); // inventoried flag
of session S1 of matching tags to B

filter.StateAwareAction.setStateAwareAction(STATE_AWARE_ACTION.STATE_AWARE_ACTION_INV_B;
// not to select tags that match the criteria

reader.Actions.PreFilters.add(filter);

// It is also required to set appropriate singulation control not to

// get tags with inventoried flag B for session 1
```

Set Appropriate Singulation Controls

Now that the pre-filters are set (i.e., tags are classified into matching or non-matching criteria), the application needs to specify which tags should participate in the inventory using `reader.Config.Antennas.setSingulationControl()`. Singulation Control must be specified with respect to each antenna like pre-filters.

State-Aware Singulation

```
// Set the singulation control

SingulationControl s1_singulationControl =
reader.Config.Antennas.getSingulationControl(1);

s1_singulationControl.setSession(SESSION.SESSION_S1);

s1_singulationControl.Action.setInventoryState(INVENTORY_STATE.INVENTORY_STATE_B);

s1_singulationControl.Action.setSLFlag(SL_FLAG.SL_FLAG_DEASSERTED);

s1_singulationControl.Action.setPerformStateAwareSingulationAction(true);

reader.Config.Antennas.setSingulationControl(1, s1_singulationControl);
```

Perform Inventory or Access operation

Inventory or Access operation when performed after setting pre-filters, use the tags filtered out of pre-filters for their operation.

Using Triggers

Triggers are the conditions that should be satisfied in order to start or stop an operation (Inventory or Access Sequence). This information can be specified using TriggerInfo class. The application can also configure the Tag-Report trigger which indicates when to receive ‘n’ unique Tag-Reports from the Reader.

We have to use `Config.setStartTrigger` and `Config.setStopTrigger` APIs to set triggers on the reader.

The following are some use-cases of using TRIGGER_INFO:

- Periodic Inventory: Start inventory at a specified time for a specified duration repeatedly.

```
TriggerInfo triggerInfo = new TriggerInfo();

// start inventory on 12th of this month and 12am and runs 200 milliseconds of every 2
seconds

triggerInfo.StartTrigger.setTriggerType(START_TRIGGER_TYPE.START_TRIGGER_TYPE_PERIODI
C);

triggerInfo.StartTrigger.Periodic.setPeriod(2000); // perform inventory for 2 seconds

// start time

SYSTEMTIME startTime = new SYSTEMTIME();

startTime.Day = 6;

startTime.Month = 8;

startTime.Year = 2011;

startTime.Hour = 9;

startTime.Minute = 20;
startTime.Second = 5;

triggerInfo.StartTrigger.Periodic.StartTime = startTime;

// stop trigger

triggerInfo.StopTrigger.setTriggerType(STOP_TRIGGER_TYPE.STOP_TRIGGER_TYPE_DURATION);

triggerInfo.StopTrigger.setDurationMilliSeconds(200); // stop after 200 milliseconds

// report back all read tags after completion of one round of inventory (i.e. one
period)

triggerInfo.setTagReportTrigger(0);
```

- Perform ‘n’ Rounds of Inventory with a timeout: Start condition could be any; Stop condition is to perform ‘n’ rounds of inventory and then stop or stop inventory after the specified timeout.

```
TriggerInfo triggerInfo = new TriggerInfo();

// start inventory immediate
```

(continued on next page)

```
triggerInfo.StartTrigger.setTriggerType(START_TRIGGER_TYPE.START_TRIGGER_TYPE_IMMEDIA
TE);

// stop trigger

triggerInfo.StopTrigger.setTriggerType(STOP_TRIGGER_TYPE.STOP_TRIGGER_TYPE_N_ATTEMPTS
_WITH_TIMEOUT);

triggerInfo.StopTrigger.NumAttempts.setN((short)3); // perform 3 rounds of inventory

triggerInfo.StopTrigger.NumAttempts.setTimeout(3000); // timeout after 3 seconds

// report back all read tags after 3 rounds of inventory

triggerInfo.setTagReportTrigger(0);
```

- Read ‘n’ tags with a timeout: Start condition could be any; Stop condition is to stop after reading ‘n’ tags or stop inventory after the specified timeout.

```
TriggerInfo triggerInfo = new TriggerInfo();

// start inventory immediate

triggerInfo.StartTrigger.setTriggerType(START_TRIGGER_TYPE.START_TRIGGER_TYPE_IMMEDIA
TE);

// stop trigger

triggerInfo.StopTrigger.setTriggerType(STOP_TRIGGER_TYPE.STOP_TRIGGER_TYPE_TAG_OBSERV
ATION_WITH_TIMEOUT);

triggerInfo.StopTrigger.TagObservation.setN((short)100); // stop inventory after
reading 100 tags

triggerInfo.StopTrigger.TagObservation.setTimeout(3000); // timeout after 3 seconds

// report back all read tags after getting 100 unique tags or after 3 seconds

triggerInfo.setTagReportTrigger(0);
```

- Inventory based on hand-held trigger: Start inventory when hand-held gun/button trigger is pulled, and stop inventory when the and-held gun/button trigger is released or subject to timeout.

```
TriggerInfo triggerInfo = new TriggerInfo();

triggerInfo.StartTrigger.setTriggerType(START_TRIGGER_TYPE.START_TRIGGER_TYPE_HANDHEL
D);

// Start Inventory when the Handheld trigger is pressed

triggerInfo.StartTrigger.Handheld.setHandheldTriggerEvent(HANDHELD_TRIGGER_EVENT_TYPE
.HANDHELD_TRIGGER_PRESSED);

triggerInfo.StopTrigger.setTriggerType(STOP_TRIGGER_TYPE.STOP_TRIGGER_TYPE_HANDHELD_W
ITH_TIMEOUT);

// Stop Inventory when the Handheld trigger is released

triggerInfo.StopTrigger.Handheld.setHandheldTriggerEvent(HANDHELD_TRIGGER_EVENT_TYPE.
HANDHELD_TRIGGER_RELEASED);

triggerInfo.StopTrigger.Handheld.setHandheldTriggerTimeout(0);
```

- Set the trigger using following APIs, perform inventory and other operation which is using above set start and stop triggers

```
reader.Config.setStartTrigger(triggerInfo.StartTrigger);
reader.Config.setStopTrigger(triggerInfo.StopTrigger);

reader.Actions.Inventory.perform(null, null, null);
```

Inventory

Inventory with Triggers

There are various situations that act as conditions (triggers) for performing inventory. See *Using Triggers on page 5-19* to configure triggers.

The following shows an example of performing one round of inventory on antennas 1 and 3.

```
TriggerInfo triggerInfo = new TriggerInfo();

// perform inventory on antenna 1 & 3

short[] antennaList = new short[] { 1, 3 };

AntennaInfo antennaInfo = new AntennaInfo(antennaList);

// start inventory immediate

triggerInfo.StartTrigger.setTriggerType(START_TRIGGER_TYPE.START_TRIGGER_TYPE_IMMEDIATE);

// stop trigger

triggerInfo.StopTrigger.setTriggerType(STOP_TRIGGER_TYPE.STOP_TRIGGER_TYPE_N_ATTEMPTS_WITH
_TIMEOUT);

triggerInfo.StopTrigger.NumAttempts.setN((short)1); // perform 1 round of inventory


triggerInfo.StopTrigger.NumAttempts.setTimeout(0); // reader default timeout

// report back all read tags after 1 round of inventory

triggerInfo.setTagReportTrigger(0);

// perform inventory

reader.Config.setStartTrigger(triggerInfo.StartTrigger);
reader.Config.setStopTrigger(triggerInfo.StopTrigger);

reader.Actions.Inventory.perform(null, null, antennaInfo);
```

Access

Using Access-Filters

In order to perform an access operation on multiple tags, the application can set ACCESS_FILTER to filter the required tags. If ACCESS_FILTER is not specified, the operation is performed on all tags. In any case, the PRE_FILTER(s) (if any is set) applies prior to ACCESS_FILTER.

The following access-filter gets all tags that have zeroed reserved memory bank.

```
AccessFilter accessFilter = new AccessFilter();

byte[] tagMask = new byte[] { (byte)0xff, (byte)0xff, (byte)0xff, (byte)0xff, (byte)0xff,
(byte)0xff, (byte)0xff, (byte)0xff };

// Tag Pattern A

accessFilter.TagPatternA.setMemoryBank(MEMORY_BANK.MEMORY_BANK_RESERVED);

accessFilter.TagPatternA.setTagPattern(new byte[] { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00 });

accessFilter.TagPatternA.setTagPatternBitCount(8 * 8);

accessFilter.TagPatternA.setBitOffset(0);

accessFilter.TagPatternA.setTagMask(tagMask);

accessFilter.TagPatternA.setTagMaskBitCount(tagMask.length * 8);

accessFilter.setAccessFilterMatchPattern(FILTER_MATCH_PATTERN.A);
```

Access Operation on Multiple Tags

Performing a single Access operation on multiple tags is an asynchronous operation. The function issues the access-operation and returns. The reader performs one round of inventory using pre-filters, if any, and then applies the access-filters and the resultant tags are subject to the access-operation. When the access operation is complete, the Dll signals the `eventStatusNotify` event with event data as INVENTORY_STOP_EVENT. In case of Read access operation (`reader.Actions.TagAccess.readEvent`) the event `eventReadNotify` is signaled when tags are reported.

The following snippet shows a sample write-access operation:

```
// Create Event to signify access operation complete

reader.Events.setInventoryStartEvent(true);

reader.Events.setInventoryStopEvent(true);

// Data Read Notification from the reader

class EventHandler implements RfidEventsListener {
```

(continued on next page)

```
          // Read Event Notification
        public void eventReadNotify(RfidReadEvents e){
            TagData tag = e.getReadEventData().tagData;
            System.out.println("Tag ID " + tag.getTagID());
            if (tag.getOpCode() == ACCESS_OPERATION_CODE.ACCESS_OPERATION_READ &&
                    tag.getOpStatus() == ACCESS_OPERATION_STATUS.ACCESS_SUCCESS)
            {
   if (tag.getMemoryBankData().length() > 0) {
                    System.out.println(" Mem Bank Data " + tag.getMemoryBankData());
                }
            }
        }
        // Status Event Notification
        public void eventStatusNotify(RfidStatusEvents e) {
            if (e.StatusEventData.getStatusEventType() ==
STATUS_EVENT_TYPE.INVENTORY_START_EVENT) {
                // Access operation started
            }
else if(e.StatusEventData.getStatusEventType() == STATUS_EVENT_TYPE.INVENTORY_STOP_EVENT)
{
// Access operation stopped - Can be used to signal waiting thread
            }
        }
    }
// Access Filter - EPC ID starting with 0x1122
AccessFilter accessFilter = new AccessFilter();
byte[] tagMask = new byte[] { 0xff, 0xff };
```

(continued on next page)

```
// Tag Pattern A
accessFilter.TagPatternA.setMemoryBank(MEMORY_BANK.MEMORY_BANK_EPC);
accessFilter.TagPatternA.setTagPattern(new byte[] { 0x11, 0x22});
accessFilter.TagPatternA.setTagPatternBitCount(2 * 8);
accessFilter.TagPatternA.setBitOffset(0);
accessFilter.TagPatternA.setTagMask(tagMask);
accessFilter.TagPatternA.setTagMaskBitCount(tagMask.length * 8);

accessFilter.setAccessFilterMatchPattern(FILTER_MATCH_PATTERN.A);

// Write user memory bank data
TagAccess tagAccess = new TagAccess();
TagAccess.WriteAccessParams writeAccessParams = tagAccess.new WriteAccessParams();
byte[] writeData = new byte[4] { 0xff, 0xff, 0xff, 0xff };
writeAccessParams.setAccessPassword(0);
writeAccessParams.setMemoryBank(MEMORY_BANK.MEMORY_BANK_USER);
writeAccessParams.setByteOffset(0);
writeAccessParams.setWriteDataLength(writeData.length);
writeAccessParams.setWriteData(writeData);
// Asynchronous write operation
reader.Actions.TagAccess.writeEvent(writeAccessParams, accessFilter, null);
// wait for access operation to complete (INVENTORY_STOP_EVENT is signaled after completing
the access operation in the eventStatusNotify)
```

Using Access Sequence

The application can issue multiple access operations on a single go using Access-Sequence API. This is useful when each tag from a set of (access-filtered) tags is to be subject to an order of access operations.

The maximum number of access-operations that can be specified in an access sequence is available in `reader.ReaderCapabilites.getMaxNumOperationsInAccessSequence` of ReaderCapabilities class.

The operations are performed in the same order in which it is added to it sequence. An operation can be removed from the sequence using reader.`Actions.TagAccess.OperationSequence.delete` and finally de-initialized if no more needed by calling the function `reader.Actions.TagAccess.OperationSequence.deleteAll()`.

```
// add Write Access operation - Write to User memory

TagAccess tagAccess = new TagAccess();

TagAccess.Sequence opSequence = tagAccess.new Sequence(tagAccess);

TagAccess.Sequence.Operation op1 = opSequence.new Operation();

op1.setAccessOperationCode(ACCESS_OPERATION_CODE.ACCESS_OPERATION_WRITE);

op1.WriteAccessParams.setMemoryBank(MEMORY_BANK.MEMORY_BANK_USER);

op1.WriteAccessParams.setAccessPassword(0);

op1.WriteAccessParams.setByteOffset(0);

op1.WriteAccessParams.setWriteData(new byte[] { (byte)0x55, (byte)0x66, (byte)0x77,
(byte)0x88 });

op1.WriteAccessParams.setWriteDataLength(4);

reader.Actions.TagAccess.OperationSequence.add(op1);

// add Write Access operation - Write to Reserved memory bank

TagAccess.Sequence.Operation op2 = opSequence.new Operation();

op2.setAccessOperationCode(ACCESS_OPERATION_CODE.ACCESS_OPERATION_WRITE);

op2.WriteAccessParams.setMemoryBank(MEMORY_BANK.MEMORY_BANK_USER);

op2.WriteAccessParams.setAccessPassword(0);

op2.WriteAccessParams.setByteOffset(0);

op2.WriteAccessParams.setWriteData(new byte[] { (byte)0xBB, (byte)0xBB, (byte)0xBB,
(byte)0xBB });

op2.WriteAccessParams.setWriteDataLength(4);

reader.Actions.TagAccess.OperationSequence.add(op2);

// perform access sequence

reader.Actions.TagAccess.OperationSequence.performSequence();

// if the access operation is to be terminated without meeting stop trigger (if specified),
stopSequence method can be called

reader.Actions.TagAccess.OperationSequence.stopSequence();
```

Gen2v2 Operations

This section covers the Gen2V2 operations that an application would need to performed on a RFID Reader which supports Gen2v2 commands such as authenticate, untraceable, and readbuffer.

authenticate

Authenticate operation takes in the message data and message length with few of the options such as decision on including the response length, sending the response, etc.

The AuthenticateParams contain the message data, message length and other settings to be sent to the reader. The accessfilter parameter contains the tag pattern on which the operation occurs.

```
// authenticate

// Tag Pattern A
accessFilter.TagPatternA.setMemoryBank(MEMORY_BANK.MEMORY_BANK_EPC);

accessFilter.TagPatternA.setTagPattern(new byte[]{(byte)0xe2, (byte)0xc0 });

accessFilter.TagPatternA.setTagPatternBitCount(16);

accessFilter.TagPatternA.setBitOffset(32);

accessFilter.TagPatternA.setTagMask(tagMask);

accessFilter.TagPatternA.setTagMaskBitCount(tagMask.length*8);

accessFilter.setAccessFilterMatchPattern(FILTER_MATCH_PATTERN.A);

Gen2v2 gen2V2 = new Gen2v2();

Gen2v2.AuthenticateParams AuthenticateParams = gen2V2.new AuthenticateParams();

AuthenticateParams.setMsgData("2001FD5D8048F48DD09AAD22000111");

AuthenticateParams.setMsgLen(120);

AuthenticateParams.setIncrespLen(true);

AuthenticateParams.setStoreResp(false);

AuthenticateParams.setSentResp(true);

try {

    reader.Actions.gen2v2Access.authenticate(AuthenticateParams, accessFilter, null);

} catch (InvalidUsageException e) {

          e.printStackTrace();

} catch (OperationFailureException e) {

e.printStackTrace();
}

)

// Keep getting response in the eventReadNotify event if registered
```

The response and result in the Tagdata will contain the information obtained from the operation.

```
public class EventHandler implements RfidEventsListener {

    // Read Event Notification

public void eventReadNotify(RfidReadEvents e) {

    TagData[] myTags = reader.Actions.getReadTags(100);

    if (myTags != null) {

        for (int index = 0; index < myTags.length; index++) {

            System.out.println("Tag ID " + myTags[index].getTagID());

            if ((myTags[index].getG2v2OpStatus()!= null) &&
(myTags[index].getG2v2OpStatus() == GEN2V2_OPERATION_STATUS.ACCESS_SUCCESS)) {

                if(!myTags[index].getG2v2Response().isEmpty()){

                    System.out.println("Gen2v2 authenticate response " +
myTags[index].getG2v2Response());
                }
            }
        }
    }
}


    // Status Event Notification

    public void eventStatusNotify(RfidStatusEvents e) {

        System.out.println("Status Notification: " +
e.StatusEventData.getStatusEventType());

        if (e.StatusEventData.getStatusEventType() ==
STATUS_EVENT_TYPE.INVENTORY_START_EVENT) {

            // Access operation started
        }
        else if(e.StatusEventData.getStatusEventType() ==
STATUS_EVENT_TYPE.INVENTORY_STOP_EVENT) {

            // Access operation stopped - Can be used to signal waiting thread
        }

    }
}
```

untraceable

Untraceable operation lets the user decide which memory bank to show and what length of the memory bank to show.

Here the UntraceableParams contain the settings and password. The accessfilter parameter contains the tag pattern on which the operation occurs.

```
// untraceable

     AccessFilter accessFilter = new AccessFilter();
  byte[] tagMask = new byte[]{(byte) 0xff, (byte) 0xff,
  };

  // Tag Pattern A
  accessFilter.TagPatternA.setMemoryBank(MEMORY_BANK.MEMORY_BANK_EPC);

  accessFilter.TagPatternA.setTagPattern("2f22");

  accessFilter.TagPatternA.setTagPatternBitCount(32);

  accessFilter.TagPatternA.setBitOffset(32);

  accessFilter.TagPatternA.setTagMask(tagMask);

  accessFilter.TagPatternA.setTagMaskBitCount(tagMask.length*8);

  accessFilter.setAccessFilterMatchPattern(FILTER_MATCH_PATTERN.A);

  Gen2v2 gen2V2 = new Gen2v2();

  Gen2v2.UntraceableParams UntraceableParams = gen2V2.new UntraceableParams();

  UntraceableParams.setPassword(0);

  UntraceableParams.setShowEpc(true);

  UntraceableParams.setHideEpc(false);

  UntraceableParams.setShowUser(false);

  UntraceableParams.setEpcLen(6);

  UntraceableParams.setTid(UNTRACEABLE_TID.HIDE_ALL_TID);

  try {

      reader.Actions.gen2v2Access.untraceable(UntraceableParams, accessFilter, null);

  } catch (InvalidUsageException e) {

      e.printStackTrace();

  } catch (OperationFailureException e) {

      e.printStackTrace();
  }
```

After this if we run the inventory we get to see the effects of the settings sent in Untraceable operation.

Tag Locationing

Readers that support the Tag Locationing feature report the same in the field isTagLocationingSupported of ReaderCapabilities as true. This feature is supported only on hand-held readers and is useful to locate a specific tag in the field of view of the reader's antenna. The default locationing algorithm supported on the reader can perform locationing only on a single antenna. reader.Actions.TagLocationing.Perform can be used to start locating a tag, and reader.Actions.TagLocationing.Stop to stop the locationing operation. The result of locationing of a tag is reported as LocationInfo in TagData and is present in TagData if tagData isContainsLocationInfo is true. tagData.LocationInfo.getRelativeDistance gives the relative distance of the tag from the reader antenna.

```
// Performing Tag Locationing on a particular tag ID.

reader.Actions.TagLocationing.Perform("E200284949150242102 0B330",null);
try {
    Thread.sleep(5000);
} catch (InterruptedException e) {
    e.printStackTrace();
}
reader.Actions.Inventory.stop();
```

The response of the tag locationing comes through eventReadNotify in the following Event Handler.

```
public class EventHandler implements RfidEventsListener {
    // Read Event Notification
    public void eventReadNotify(RfidReadEvents e) {
        TagData[] myTags = reader.Actions.getReadTags(100);
        if (myTags != null) {
            for (int index = 0; index < myTags.length; index++) {
                System.out.println("Tag ID " + myTags[index].getTagID());
                if (myTags[index].isContainsLocationInfo()) {
                    int tag = index;
                    System.out.println("Tag locationing distance " +
myTags[tag].LocationInfo.getRelativeDistance());
                }
            }
        }
    }
}
```

Exceptions

The Zebra RFID Android SDK throws two types of exceptions as a given:

- InvalidUsageException: This exception is thrown when the user passes an invalid parameter, calling getInfo() gives detail error message.
- OperationFailureException: This exception is thrown when the requested operation is failed. The Exception contains the Operation RFIDResults, status description, time stamp & vendor specific message for the operation failure.

Exception Handling

All API should be called under try-catch block to catch the exception thrown while performing API by SDK.

```
try {
    rfidReader.connect();
} catch (InvalidUsageException e) {
    e.printStackTrace();
} catch (OperationFailureException e) {
    e.printStackTrace();
    ex = e;
}
```

Chapter 6 ZETI PROGRAMMING GUIDE

ZETI Prerequisites

Before using ZETI, ensure the following steps are completed.

- Pair the RFD8500 with a Bluetooth device (see *Pairing with Bluetooth on page 1-2*).
- Open a Bluetooth serial communication port using an API supported by the platform. If the platform is Windows PC, open a COM port via terminal application (see *Using a PC Based Terminal Over ZETI with the RFD8500 on page 1-9*).
- Establish a ZETI connection by sending a 'connect' or 'cn' command to the RFD8500 over the Bluetooth serial port opened in the previous step. A successful connection message displays (Command: Connect; Status: Connection Successful).

ZETI Format

Command strings are in the following format:

Command name (or its two letter abbreviation) > zero or additional optional parameters, where each parameter is preceded by a dot character (**.**) > parameter id string (or parameter id abbreviation) > corresponding parameter value (each separated by a space character) > <CR><LF>.

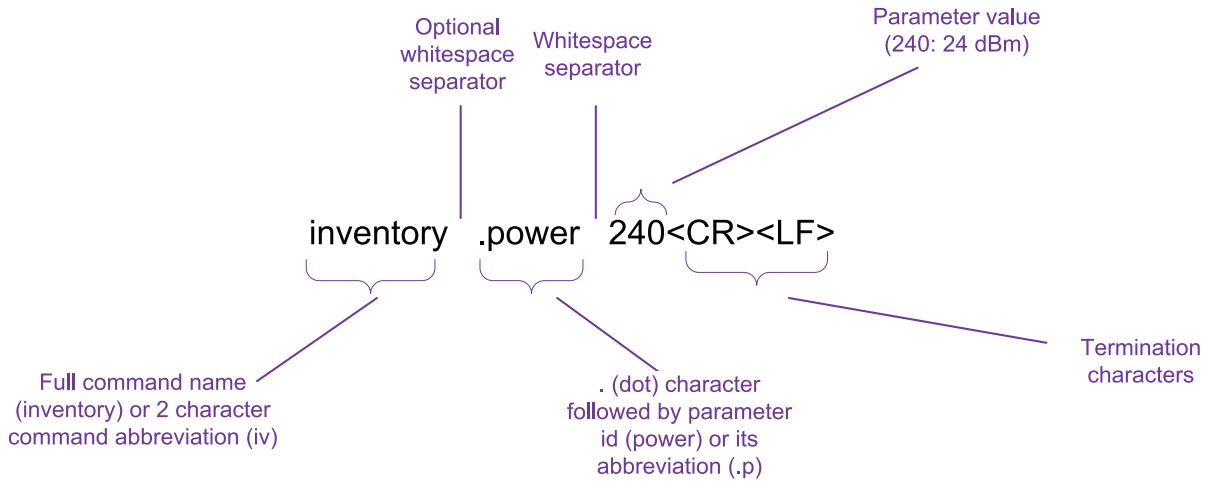

Figure 6-1 *Example Inventory Command - "Inventory .power 240<CR><LF>"*

By default, the response for the command are lines of metadata ending with <CR><LF>. Each line of metadata includes the fields reported, each comma separated. The end of response is indicated by a single line that only includes <CR><LF>. To optimize reported response size, when multiple lines of data are present in a response, fields that do not change from prior line are blank.

Example response for the inventory command in *Figure 6-1 on page 6-2*:

```
Command:inventory,Status:0,EPC:,RSSI:,TS:<CR><LF>
,,32001122334455667788901,-45,22334455<CR><LF>
,,32001122334455667788912,-56,<CR><LF>
,,32001122334455667788903,-60,<CR><LF>
,,32001122334455667788904,-44,22334465<CR><LF>
<CR><LF>
```

Figure 6-2 illustrates the components of the response for a command.

Use the *abort* command to stop operation (e.g., abort <CR><LF>).

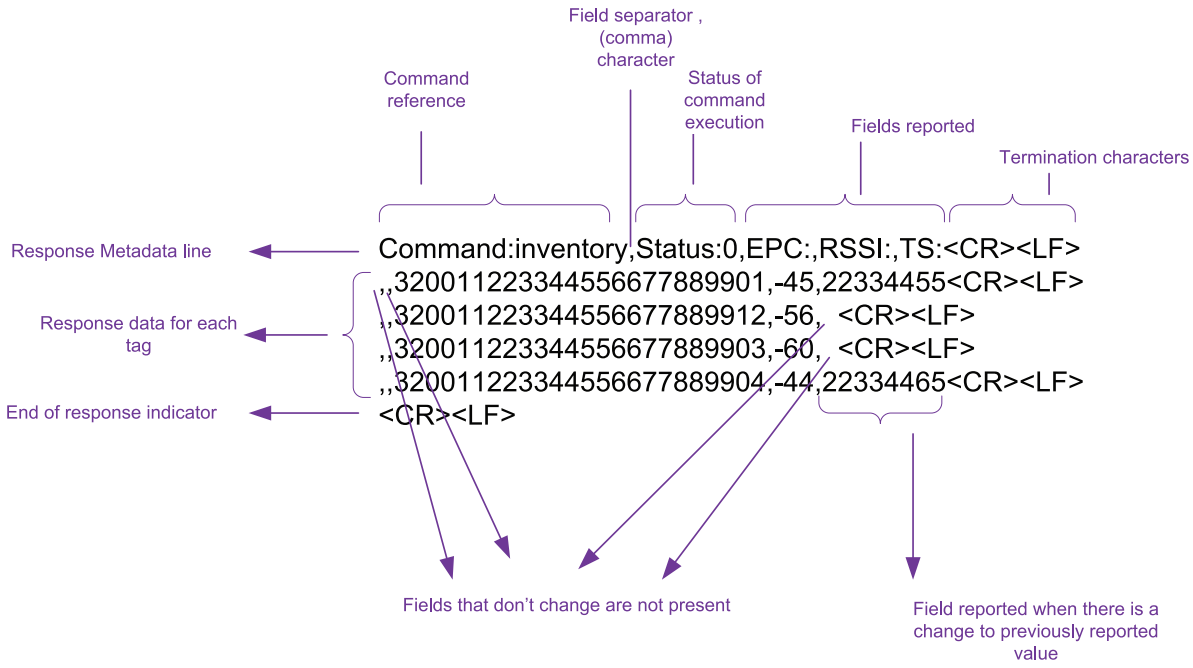

Figure 6-2 *Components of the Response Command*

For more information see *Appendix A, ZETI REFERENCE*.

General Flow.

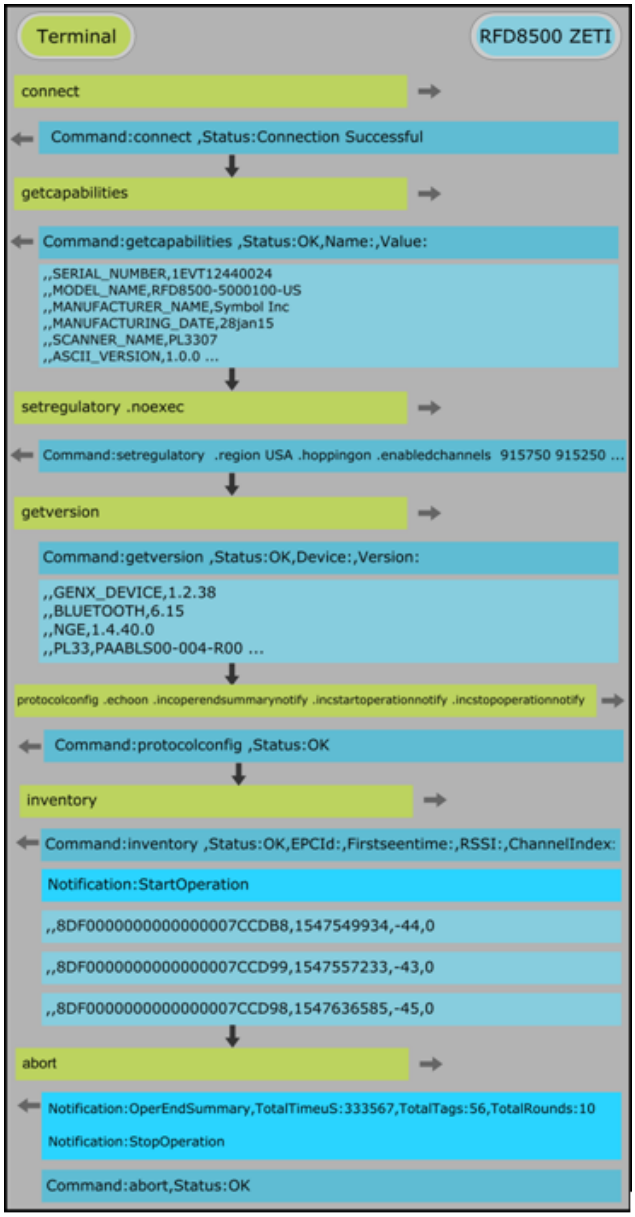

Figure 6-3 *Components of the Response Command*

NOTE Metadata/status, data and asynchronous notifications are marked with different color shade.

Getting the Reader Capabilities

The capabilities of the reader can be accessed using the *getcapabilities* or *gc* command.

The reader capabilities include *serial/MAC numbers*, *model name*, *version* information, and the maximum allowed number of supported features like filters and power.

Example command:

```
gc
Command:getcapabilities ,Status:OK,Name:,Value:
,,SERIAL_NUMBER,1EVT12440024
,,MODEL_NAME,RFD8500-5000100-US
,,MANUFACTURER_NAME,Zebra Tech Inc
,,MANUFACTURING_DATE,28jan15
,,SCANNER_NAME,PL3307
,,ASCII_VERSION,1.0.0
,,SELECT_FILTERS,4
,,MIN_POWER,120
,,MAX_POWER,300
,,POWER_STEPS,1
,,AIR_PROTOCOL_VERSION,1.2.0
,,MAX_ACCESS_SEQUENCE,10
,,BD_ADDRESS,0017E970AAB1
```

Configuring the Reader

Antenna Configuration

Using a *help* command returns all supported ZETI commands. Using the *help* command with a supported ZETI command returns more detail. Note the short form of the command in the parenthesis.

```
help ac
Command:setantennaconfiguration(ac) Parameter            Range
                                    power(p)             120 to 300
                                    linkprofileindex(lx) 0 to 35
                                    tari(ta)             0 to 4294967295
                                    doselect(ds)
                                    noselect(ns)
                                    defaults(d)
                                    noexec(n)
```

Examples:

- To configure antenna power to 2700 dBm, and link profile zero:
  ```
  ac .p 270 .lx 0
  Command:setantennaconfiguration ,Status:OK
  ```
- By using no execute option (.n) the current configuration can be retrieved:
  ```
  ac .n
  Command:setantennaconfiguration  .power 240 .linkprofileindex 0 .tari 0
  .noselect .noexec:1,Status:OK
  ```
- By using the default (.d) you can revert to default settings:
  ```
  ac .d
  Command:setantennaconfiguration ,Status:OK
  ```

Report Configuration

Reports can be configured using the command *setreportconfig* or *rc*. The command includes options to enable/disable reporting tag metadata such as *seen time*, *RSSI*, *Phase*, etc.

For example, to include *RSSI* use *ir*, and to exclude *Phase* use *ek* (if *Phase* was previously included).

```
help rc
Command:setreportconfig(rc)          Parameter                Range
                                     incfirstseentime(iz)
                                     excfirstseentime(ez)
                                     inclastseentime(il)
                                     exclastseentime(el)
                                     incpc(ic)
                                     excpc(ec)
                                     incrssi(ir)
                                     excrssi(er)
                                     incphase(ik)
                                     excphase(ek)
                                     incchannelindex(ih)
                                     excchannelindex(eh)
                                     inctagseencount(is)
                                     exctagseencount(es)
                                     defaults(d)
                                     noexec(n)
```

Examples:

- Following command enables firstseentime, RSSI & Channel index in report configuration:
  ```
  rc .iz .ir .ih
  Command:setreportconfig ,Status:OK
  ```
- By using default (.d) and no execute (.n) options together default values can be seen without setting them.
  ```
  rc .d .n
  Command:setreportconfig  .incfirstseentime .exclastseentime .excpc .incrssi
  .excphase .excchannelindex .exctagseencount .noexec:1,Status:OK
  ```

Beeper Configuration

The beeper is configured using the set attribute command *setattr*, where attvalue 0 represents high volume (use 1 for medium and 2 for low).

Example:

```
setattr .attnum 140 .atttype B .attvalue 0
Command:setattr,Status:OK
```

Simple Inventory and Abort

Current inventory configurations are retrieved by using the (.n) option. Power and Report configurations set in earlier steps are reflected in the Inventory operation.

The Inventory command is *inventory* or *in*.

```
in .n
Command:inventory  .batchmode auto .incfirstseentime .exclastseentime .excpc
.incrssi .excphase .incchannelindex .exctagseencount .doselect .power 270
.noexec:1,Status:OK
```

To start inventory enter *in*. To stop inventory enter *abort* or *a*.

Example:

```
in
Command:inventory ,Status:OK,EPCId:,Firstseentime:,RSSI:,ChannelIndex:
,,8DF0000000000000007CCDB8,1547549934,-44,0
,,8DF0000000000000007CCD99,1547557233,-43,0
,,8DF0000000000000007CCDA8,1547561094,-43,0
,,8DF0000000000000007CCD98,1547636585,-45,0
a
Command:abort,Status:OK
```

Handling Tags, Events and Start/Stop Notifications

Events and notifications can be enabled using the command *protocolconfig* or *sa*.

protocolconfig includes notifications such as, *operation summary*, *inventory summary*, *start/stop*, *trigger*, *battery related* events, and others.

Example commands:

```
incoperendsummarynotify(io) incinvendsummarynotify(ii)
incstartoperationnotify(is)
incstopoperationnotify(it), inctriggereventnotify(ig) & incbatteryeventnotify(ib)
sa .io .ii .is .it .ig .ib
Command:protocolconfig ,Status:OK
```

The commands *setstarttrigger(st)* and *setstoptrigger(ot)* are used to set the physical trigger conditions for operation in order to see trigger related notifications.

The commands *startonhandheldtrigger(sp)* and *triggertype(tt)* are set to zero (press).

```
st .sp .tt 0
Command:setstarttrigger ,Status:OK
```

(continued on next page)

The commands *stoponhandheldtrigger(tp)*, *triggertype(tt)* are to one (release), and 5 second *stoptimeout (to)*.

```
ot .tp .tt 1 .to 5000
Command:setstoptrigger ,Status:OK
```

The following Inventory command response shows all enabled notifications.

```
in
Command:inventory ,Status:OK,EPCId:,Firstseentime:,RSSI:,ChannelIndex:
Notification:TriggerEvent,TriggerValue:0
Notification:StartOperation
,,8DF0000000000000007CCDBD,146569510,-40,4
,,8DF0000000000000007CCD8E,146605458,-41,4
,,8DF0000000000000007CCDAE,146618516,-43,4
,,8DF0000000000000007CCD7C,146624783,-42,4
,,8DF0000000000000007CCDD7,146643587,-38,4
,,000000000000000000000253,146647432,-38,4
Notification:TriggerEvent,TriggerValue:1
Notification:OperEndSummary,TotalTimeuS:1197949,TotalTags:30,TotalRounds:4
Notification:StopOperation
```

The following incoming notifications show connecting and removing the charge to the RFD8500, respectively.

```
Notification:BatteryEvent,Cause:Charger is Connected,Level:53,Charging:true
Notification:BatteryEvent,Cause:Charger is Disconnected,Level:60,Charging:false
```

Simple Access Operation (Read, Write, Lock, Kill)

The '.d' option can be used to restore all configurations to their default state. Below, default conditions were set to protocolconfig, setreportconfig, start and stop triggers (which were modified in earlier steps).

```
sa .d
Command:protocolconfig ,Status:OK
rc .d
Command:setreportconfig ,Status:OK
st .d
Command:setstarttrigger ,Status:OK
ot .d
Command:setstoptrigger ,Status:OK
```

The stop trigger condition is now set to do a single access operation.

```
ot .ea .sa 1
Command:setstoptrigger ,Status:OK
```

Read access operation on user memory bank (default) can be done using following:

```
rd
Command:read ,Status:OK,EPCId:,Firstseentime:,RSSI:,readStatus:,user:
,,E2002849491502351020B318,1830102418,-33,,000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000
```

(continued on next page)

The following exemplary example sets the access password (reserved memory bank), locking the user memory bank for a write operation:

```
wr .b reserved .f 2 .x AABBCCDD
Command:write ,Status:OK,EPCId:,Firstseentime:,RSSI:,writeStatus:,NumWritten:
,,E200284949150235102 0B318,1912599493,-32,,2
lo .w AABBCCDD .usermem 2
Command:lock ,Status:OK,EPCId:,Firstseentime:,RSSI:,lockStatus:
,,E200284949150235102 0B318,1942830592,-33,
```

Trying to write in a locked area results in error:

```
wr .f 0 .x 11223344
Command:write ,Status:OK,EPCId:,Firstseentime:,RSSI:,writeStatus:,NumWritten:
,,E200284949150235102 0B318,1967721994,-30,Tag Locked Error,0
```

A successful write and read, using the password. Note that the number of successful bytes written is two:

```
wr .f 0 .x 11223344 .w AABBCCDD
Command:write ,Status:OK,EPCId:,Firstseentime:,RSSI:,writeStatus:,NumWritten:
,,E200284949150235102 0B318,1977931276,-30,,2
rd
Command:read ,Status:OK,EPCId:,Firstseentime:,RSSI:,readStatus:,user:
,,E200284949150235102 0B318,1984136665,-30,,1122334400000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000
```

The following tag kill example in the first command shows existing kill password in reserved memory area and then tag kill operation is performed.

```
rd .b reserved
Command:read ,Status:OK,EPCId:,Firstseentime:,RSSI:,readStatus:,reserved:
,,8DF000000000000812E39,2279812125,-35,,1122334400000000
kl .w 11223344
Command:kill ,Status:OK,EPCId:,Firstseentime:,RSSI:,killStatus:
,,8DF000000000000812E39,2298543322,-37,
```

As kill response status shows no error, one can safely trash the killed tag at this time.

Advanced Operations

This section describes some of the advanced operations that can be performed on RFD8500 using ZETI.

Using Pre-Filters

Pre-filters, or select, can be performed using setselectrecords and setqueryparams combinations. Before moving on, you can set the default start and stop after five tag counts.

```
ot .ec .tc 5
Command:setstoptrigger ,Status:OK
```

Select record set with target as s0, action to move matching tag to inv a and non matching to inv b state, operated memory bank epc, start positions to 32nd bit and match length to 16 bits.

```
sr .t .g 0 .o 0 .q epc .a 32 .m 8DF0 .l 16
Command:setselectrecords ,Status:OK
Query parameter is set to select ALL, session 0, and target inv state A
qp .e 0 .i 0 .j 0
Command:setqueryparams ,Status:OK
Check configuration done above:
sr .n
Command:setselectrecords  .selectrecord .target 0 .action 0 .maskbank epc
.maskstartpos 32 .matchpattern 8DF0 .matchlength 16 .notruncate
.noexec:1,Status:OK
qp .n
Command:setqueryparams  .queryselect 0 .querysession 0 .querytarget 0 .population
30 .noexec:1,Status:OK
Inventory operation started with select record applied, result shows matching tags
in .doselect
Command:inventory ,Status:OK,EPCId:,Firstseentime:,RSSI:
,,8DF000000000000000812E3A,396294563,-36
,,8DF000000000000000812E3B,396300563,-38
,,8DF000000000000000812E3A,396319630,-36
,,8DF000000000000000812E3B,396323479,-38
```

Query parameter changed to target inv state B flag (non matching tags moved to inv b).

```
qp .j 1
Command:setqueryparams ,Status:OK
Inventory operation started with select record applied, result shows non matching
tags
in .doselect
Command:inventory ,Status:OK,EPCId:,Firstseentime:,RSSI:
,,000000000000000000000253,563037828,-36
,,000000000000000000000253,567803654,-38
,,000000000000000000000252,567812289,-38
,,000000000000000000000252,567829605,-38
```

Using Start and Stop Triggers

Inventory starts after a 1000ms delay, reads 10 tags, another 1000ms delay, reads 10 tags. This continues until an abort is given. Notifications were enabled to see when the operation rounds start and end.

```
st .ro .sd 1000
Command:setstarttrigger ,Status:OK
ot .ec .tc 2
Command:setstoptrigger ,Status:OK
in
Command:inventory ,Status:OK,EPCId:,Firstseentime:,RSSI:
,,00000000000000000000253,859739100,-37
,,8DF000000000000812E3A,864135617,-36
Notification:OperEndSummary,TotalTimeuS:51742,TotalTags:2,TotalRounds:2
,,00000000000000000000253,864162904,-37
,,8DF000000000000812E3B,865156101,-36
Notification:OperEndSummary,TotalTimeuS:62170,TotalTags:2,TotalRounds:2
,,00000000000000000000252,865161345,-37
,,8DF000000000000812E3B,866109329,-38
Notification:OperEndSummary,TotalTimeuS:23429,TotalTags:2,TotalRounds:2
a

Command:abort,Status:OK

Inventory starts on pressing the trigger and stops after 10 inventory rounds.
st .sp .tt 0
ot .ei .si 10
Command:setstoptrigger ,Status:OK
in
Command:inventory ,Status:OK,EPCId:,Firstseentime:,RSSI:
Notification:StartOperation
,,00000000000000000000252,1092359947,-38
,,00000000000000000000253,1341606260,-37
,,00000000000000000000252,1341610687,-38
,,000000000000000000000AD,1341615370,-45
,,00000000000000000000252,1341622204,-38
,,00000000000000000000253,1341628860,-37
Notification:OperEndSummary,TotalTimeuS:203380,TotalTags:29,TotalRounds:10
Notification:StopOperation
```

The Read command begins reading tags on releasing the trigger and continues until an abort is given. On pressing the trigger, it stops reading tags, and again on release it starts reading tags again continuously until an abort is given.

```
st .sp .tt 1 .ro
Command:setstarttrigger ,Status:OK
ot .d
Command:setstoptrigger ,Status:OK
ot .ea .sa 4
Command:setstoptrigger ,Status:OK
rd
Command:read ,Status:OK,EPCId:,Firstseentime:,RSSI:,readStatus:,user:
```

(continued on next page)

```
Notification:StartOperation
,,000000000000000000000253,1479045271,-37,,00000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000
,,000000000000000000000252,1479065972,-37,,00000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000
,,000000000000000000000252,1479097173,-37,,00000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000
,,000000000000000000000253,1479121453,-37,Read Length Error
Notification:OperEndSummary,TotalTimeuS:168573,TotalTags:5,TotalRounds:3
Notification:StopOperation
Notification:StartOperation
,,000000000000000000000253,1479541918,-37,,00000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000
,,0000000000000000000000AD,1479562109,-44,Tag Response CRC Error
,,000000000000000000000252,1479575428,-38,,00000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000
,,000000000000000000000253,1479600958,-37,,00000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000
Notification:OperEndSummary,TotalTimeuS:96305,TotalTags:4,TotalRounds:2
Notification:StopOperation
Notification:StartOperation
,,000000000000000000000253,1479636357,-37,,00000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000
,,000000000000000000000252,1479653635,-38,,00000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000
,,000000000000000000000253,1479682745,-37,,00000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000
,,000000000000000000000252,1479702787,-38,,00000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000
Notification:OperEndSummary,TotalTimeuS:96437,TotalTags:4,TotalRounds:3
Notification:StopOperation
Notification:StartOperation
a,,000000000000000000000252,1480247965,-37,,0000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000
,,000000000000000000000253,1480265402,-37,,00000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000

Notification:OperEndSummary,TotalTimeuS:144029,TotalTags:3,TotalRounds:3

Notification:StopOperation
Command:abort,Status:OK
```

Access with Access Criteria

We can define access criteria using the setaccesscriteria command. The following example shows first access criteria on EPC memory bank word offset set to two, length one word, and mask as 0xFFFF.

```
at .c .q1 epc .a1 2 .l1 1 .d1 E200 .m1 FFFF .o1
Command:setaccesscriteria ,Status:OK
at .accesscriteria .filter1maskbank epc .filter1maskstartpos 2
.filter1matchlength 1 .filter1data E200 .filter1mask FFFF .filter1domatch
Command:setaccesscriteria ,Status:OK
```

Set stop trigger to do one access operation only.

```
ot .ea .sa 1
Command:setstoptrigger ,Status:OK
```

Read access operation with first access criteria being applied.

```
rd .ci 1 .b epc .f 2 .h 6
Command:read ,Status:OK,EPCId:,Firstseentime:,RSSI:,readStatus:,epc:
,,E200284949150236102 0B31C,1031167844,-32,,E200284949150236102 0B31C
```

Access Sequence

The following section describes usage of the access sequence, where up to ten access commands can be pipelined. First access stop count is set to two so that two rounds of an access sequence completes.

```
ot .ea .sa 2
Command:setstoptrigger ,Status:OK
```

Begin access sequence 'ba' command, then read reserved memory bank, write access password, lock access password, and read again. At the end, 'ea' completes access sequence.

```
ba
Command:beginaccesssequence ,Status:OK
rd .ci 1 .b reserved .f 2 .h 2
Command:read ,Status:OK,CmdNum:1
wr .ci 1 .b reserved .f 2 .x AABBCCDD
Command:write ,Status:OK,CmdNum:2
lo .ci 1 .w AABBCCDD .ap 2
Command:lock ,Status:OK,CmdNum:3
rd .ci 1 .b reserved .f 2 .h 2
Command:read ,Status:OK,CmdNum:4
ea
Command:endaccesssequence ,Status:OK
```

Execaccesssequence, or 'xa', starts performing stored access sequence. The meta data structure can be seen matching with commands in sequence, and successful result. When the second round is started the first read operation fails as memory bank is locked and access sequence is aborted.

```
xa
Command:execaccesssequence
,Status:OK,EPCId:,Firstseentime:,RSSI:,readStatus:,reserved:,writeStatus:,NumWrit
ten:,lockStatus:,readStatus:,reserved:
,,E200284949150236102 0B31C,5758908545,-30,,00000000,,2,,,AABBCCDD
,,E200284949150236102 0B31C,5758968197,-30,Tag Locked Error
```

NXP Gen2V2 Features

Following section shows various Gen2v2 related operation, it requires NXP DNA tag with key0 and key1 programmed.

Authenticate command with challenge and access criteria set to specific tag, response is AES encrypted challenge included.

au .sr .il .l 96 .d 000096564402375796C69664 .ci 1

Command:authenticate ,Status:OK,EPCId:,Firstseentime:,RSSI:,result:,response:

Notification:StartOperation

,,E2C06F920000003A001057C7,729500887,-30,Send with Length,0080E4B3D3BEE9C898BD8C11BFD624F10079

Notification:OperEndSummary,TotalTimeuS:55943,TotalTags:2,TotalRounds:1

Notification:StopOperation

Untraceable command used to hide epc and only two word of EPC is being returned afterwards when inventory is performed.

uc .he .el 2 .ci 1

Command:untraceable ,Status:OK,EPCId:,Firstseentime:,RSSI:,result:,response:

Notification:StartOperation

,,E2C06F920000003A001057C7,796444354,-30,,

Notification:OperEndSummary,TotalTimeuS:291256,TotalTags:4,TotalRounds:2

Notification:StopOperation

in

Command:inventory ,Status:OK,EPCId:,Firstseentime:,RSSI:

Notification:StartOperation

,,E2C06F92,813606493,-32

Authenticate command with challenge and flag set to include epc; response is AES encrypted challenge and hidden epc id included.

au .sr .il .l 120 .d 2001FD5D8048F48DD09AAD22000111 .ci 1

Command:authenticate ,Status:OK,EPCId:,Firstseentime:,RSSI:,result:,response:

Notification:StartOperation

,,E2C06F92,990494932,-32,Send with Length,01006F1BF31FA3AD2271746AEDF9D6994516D474F2E9858DC2FBA0FA94CE90186EC8

Notification:OperEndSummary,TotalTimeuS:53210,TotalTags:1,TotalRounds:2

Notification:StopOperation

Use Untraceable command to unhide EPC memory bank.

uc .el 6 .ci 1

(continued on next page)

Command:untraceable ,Status:OK,EPCId:,Firstseentime:,RSSI:,result:,response:

Notification:StartOperation

,,E2C06F920000003A001057C7,903219826,-30,,

Notification:OperEndSummary,TotalTimeuS:43288,TotalTags:1,TotalRounds:2

Notification:StopOperation

in

Command:inventory ,Status:OK,EPCId:,Firstseentime:,RSSI:

Notification:StartOperation

,,E2C06F920000003A001057C7,928051239,-30

,,E2C06F920000003A001057C7,928063327,-30

Tag Locationing

Tag locating can be accomplished with the 'locatetag' command; response includes proximity of tag in percentage. The tag was far from the RFD8500 antenna, and then moved closer to the antenna.

```
lt .ep 8DF000000000000000812E3B
Command:locatetag ,Status:OK,Proximitypercent:
Notification:StartOperation
,,0
,,1
,,5
,,9
,,13
,,18
,,24
,,30
,,38
,,46
,,53
,,59
,,64
,,84
,,82
,,82
,,84
,,84
,,99
,,99
,,99
,,99
,,100
,,100
,32
a

Notification:StopOperation
Command:abort,Status:OK
```

Batch Mode and getTags, PurgeTags

The batch mode feature is used to store tag data in an internal tag database, maintained in the RFD8500, in case Bluetooth disconnects (auto batch mode) or always enable case.

Inventory stops after a five second timeout.

```
ot .et .to 5000
Command:setstoptrigger ,Status:OK
Start inventory with batch mode enable, no tag is returned to console/app.
in .bm enable
Command:inventory ,Status:Inventory Statred in Batch Mode
Notification:StartOperation
Notification:OperEndSummary,TotalTimeuS:5000232,TotalTags:295,TotalRounds:27
Notification:StopOperation
```

Get the inventoried tags using 'gettags' or 'tg' command.

```
gettags
Command:gettags ,Status:OK,EPCId:,Firstseentime:,RSSI:
,,000000000000000000000AD,1480693640,-30
,,8DF000000000000000812E3B,1480890332,-43
,,8DF000000000000000812E3F,1484287709,-44
,,8DF000000000000000812E40,1480689779,-31
,,8DF000000000000000812E41,1481292071,-41
,,8DF000000000000000812E42,1480770946,-31
,,8DF000000000000000812E44,1480685900,-36
,,8DF000000000000000812E45,1480734067,-33
,,8DF000000000000000812E46,1481069209,-45
,,8DF000000000000000812E4E,1481157669,-38
,,8DF000000000000000812E4F,1480738553,-34
,,8DF000000000000000812E50,1480678147,-32
,,8DF000000000000000812E51,1480720953,-34
,,8DF000000000000000812E52,1480801763,-35
,,8DF000000000000000812E53,1480705136,-33
,,8DF000000000000000812E54,1480682067,-34
,,8DF000000000000000812E55,1480754665,-33
,,8DF000000000000000812E57,1480791267,-39
,,8DF000000000000000812E58,1481130077,-44
,,8DF000000000000000812E5B,1480674276,-31
,,8DF000000000000000812E5D,1480749405,-33

Notification:StopOperation
```

Purge the store tag data base using 'purgetags' or 'tp' command, querying tags afterwards returns empty response.

```
tp
Command:purgetags ,Status:OK
tg
Command:gettags ,Status:OK,EPCId:,Firstseentime:,RSSI:

Notification:StopOperation
```

Management and Configuration

Setting and Getting Region Configuration

setregulatory can be used to set the regulatory settings. This command provides options to set the region along with the channels depending on whether the area is hopping configurable or not. Setting the region saves to non volatile memory automatically.

```
sg .c USA
Command:setregulatory ,Status:OK
```

getregion returns the SupportedChannels and hopping state for the region set. Additionally, getregion with a specified region code returns the information for that region code.

```
gr
Command:getregion ,Status:OK,RegionCode:,HoppingConfigurable:,SupportedChannels:
,,USA,false, 915750 915250 903250 926750 926250 904250 927250 920250 919250 909250
918750 917750 905250 904750 925250 921750 914750 906750 913750 922250 911250
911750 903750 908750 905750 912250 906250 917250 914250 907250 918250 916250
910250 910750 907750 924750 909750 919750 916750 913250 923750 908250 925750
912750 924250 921250 920750 922750 902750 923250
```

SaveConfig Including resetTodefaults

The Changeconfig command is used to change the configuration of the device. The saveconfig command saves the configuration to the parambuffer (on non volatile medium) so that it is available across the reboot. The savecustomdefaults command saves the current configuration to parambuffer and also to the custom area of the flash so that one can restore to these defaults using restorecustomdefaults. restorefactorydefaults restores the device's configuration to factory settings.

```
cc .m saveconfig
Command:changeconfig ,Status:OK
```

NOTE Pairing and reconnecting the Device/Phone is required again after resetting factory defaults. Region also requires reconfiguration.

```
cc .m restorefactorydefaults
```

Connection with Password

The device can be configured to require a password to establish a ZETI connection on top of the Bluetooth connection.

To set a ZETI connection password on the RFD8500:

```
btp .p RFID#123 .r RFID#123
Command:btpassword ,Status:OK
```

The connection command is in the following format with the password:

```
cn .p RFID#123
Command:connect ,Status:Connection Successful
```

If a connection is attempted without the password, or an incorrect password, the following error returns:

```
Command:connect,Status:Password mismatch error
```

ASCII Protocol Configuration

To enable or disable any type of notification, use the command protocolconfig with the appropriate option. This command can additionally be used to turn the echo on/off, include or exclude CRC, and to set the debuginterface.

Enable echo, CRC, operationendsummary, invendsummary using the following command.

```
sa .ee .ic .io .ii
```

Settings are reflected by the *no execute* option.

```
sa .n
Command:protocolconfig  .echoon .debuginterface 0 .disabledebug .inccrc
.incoperendsummarynotify .incinvendsummarynotify .excstartoperationnotify
.excstopoperationnotify .exctriggereventnotify .incbatteryeventnotify
.inctemperatureeventnotify .excpowereventnotify .excdatabaseeventnotify
.excradioerroreventnotify .noexec:1,Status:OK,CRC:34880
```

This can be verified using the inventory command as follows:

```
in
Command:inventory ,Status:OK,EPCId:,Firstseentime:,RSSI:,CRC:11416
,,8DF000000000000000812E53,341365137,-33,10090
,,8DF000000000000000812E4F,341369004,-38,31512
,,8DF000000000000000812E50,341372883,-32,39229
,,8DF000000000000000812E51,341376758,-34,28735
,,8DF000000000000000812E40,341382465,-30,38466
,,8DF000000000000000812E4C,341386337,-39,3186
,,000000000000000000000000AD,341390219,-30,27696
,,8DF000000000000000812E54,341394089,-35,21719
,,8DF000000000000000812E42,341397957,-30,6255
,,8DF000000000000000812E5B,341401829,-32,35067
,,8DF000000000000000812E48,341405699,-35,15732
,,8DF000000000000000812E52,341431542,-36,38840
,,8DF000000000000000812E4B,341435425,-33,52119
,,8DF000000000000000812E55,341444509,-32,48352
,,8DF000000000000000812E45,341494097,-35,50429
,,8DF000000000000000812E44,341504965,-37,19862
a
Notification:OperEndSummary,TotalTimeuS:392133,TotalTags:16,TotalRounds:1,CRC:315
38

Command:abort,Status:OK,CRC:50380
```

Use following command to exclude CRC:

```
sa  .ec
```

GetVersion

getversion returns the platform version information for RFD8500 device. It also returns the version information for Bluetooth stack, NGE(Radio), PL33 (imager) and the device's hardware.

```
gv
Command:getversion ,Status:OK,Device:,Version:
,,GENX_DEVICE,1.2.37
,,BLUETOOTH,6.15
,,NGE,1.4.40.0
,,PL33,PAABLS00-004-R00
,,HARDWARE,1
```

Battery and Device Information

Help getdeviceinfo returns all the options available with the command.

```
help gd
Command:getdeviceinfo(gd)     Parameter          Range

                              battery(bt)
                              temperature(tp)
                              power(p)
```

getdeviceinfo with any of the options (battery/temperature/power) or any combination of these options returns a notification with the relevant information about the option specified.

getdeviceinfo with battery returns the battery's charging status, battery level and the cause of the notification (which can either be a user request or any of the internal alarms).

```
gd .bt
Notification:BatteryEvent,Cause:User Request,Level:89,Charging:true
```

getdeviceinfo with temperature returns the STM32, Radio's PA temperature and the cause of the notification.

getdeviceinfo with all options returns notification for all options.

Also charger was connected afterwards causes notification with positive current.

```
gd .bt .tp .p
Notification:BatteryEvent,Cause:User Request,Level:90,Charging:false
Notification:TemperatureEvent,Cause:User Request,STM32 Temp:55,Radio PA Temp:44
Notification:PowerEvent,Cause:User Request,Voltage:4028,Current:-317,Power:-1276
Notification:BatteryEvent,Cause:Charger is Connected,Level:89,Charging:true
```

Appendix A ZETI REFERENCE

ZETI Interface Command Reference

See *Table A-3 on page A-45* for the possible errors reported back for ZETI commands.

Table A-1 *ZETI Interface Command Reference*

Command	Parameters: Parameter ID	Parameters: Options, Default Value, Range	Response	Example
Configuration Commands				

setselectrecords(sr)
 	
Description: Configure RFID Gen2 protocol select records.
Up to four selected records are supported on the device. Specified select records are applied before an Inventory operation for all RFID Air Interface operational commands, if such commands explicitly specifies doselect(ds) option.
 	
(continued on next page)
	defaults(d)	None.	One line metadata response indicates status of adding select records terminated with <CR><LF>. As no data is associated with response, metadata line is followed by another <CR><LF> indicating end of response.	Command issued to add two select records
				• First to select all tags with EPC banks having PC word as 3200
				• Then to select all tags with USER bank starting with word data as 1234:
				`setselectrecords .selectrecord .maskpattern 3200 .selectrecord .maskbank user .maskstartpos 0 .matchpattern 1234<CR><LF>`
				Response:
				`Command:setselectrecords,Status:OK<CR><LF><CR><LF>`

	selectrecord(t)	Identifies beginning of a select record. All options after this till next selectrecord(t) option or end of command indicated by <CR><LF> (in the case of last record) compose one select record.		

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters		Response	Example
	Parameter ID	Options, Default Value, Range		
setselectrecords(sr) (continued from previous page)	target(g)	One of following five ASCII integer values: 0: Session S0 1: Session S1 2: Session S2 3: Session S3 4: Select Flag (default).		
	action(o)	One of eight integer values in ASCII character format; see *Table A-2 on page A-44*.		
	maskbank(q)	One of epc(default)/tid/user		
	maskstartpos(a)	4 character ASCII hex string (default: 10). Indicates start bit position from beginning of memory bank from where match pattern is checked.		
	matchpattern(m)	ASCII hex string (default: 3000). Each character in string padded to ensure full byte.		
	matchlength(l)	2 character ASCII Hex value (default: 16). Represents numbers of bits from start of match pattern to be used for select mask.		
	dotruncate(dt)	None.		
	notruncate(nt)	None (default).		
	nonexec(n)			
setqueryparams(qp) Description: Configure parameters for RFID Gen 2 Query command. (continued on next page)	defaults(d)	None.	One line metadata response indicate status of setting the query parameters terminated with <CR><LF>. As no data is associated with response, metadata line is followed by another <CR><LF> indicating end of response.	Command issued to set query parameters with tag population of 100: `setqueryparams .y 100<CR><LF>` Response: Command: setqueryparams,Status:OK<CR><LF><CR><LF>

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters			
	Parameter ID	Options, Default Value, Range	Response	Example
setqueryparams(qp) (continued from previous page)	queryselect(e)	One of the following ASCII character values (default: 0): 0 or 1: All. 2: Select De-asserted. 3: Select Asserted.		
	querysession(i)	One of the following ASCII character values (default: 0): 0: Session S0. 1: Session S1. 2: Session S2. 3: Session S3.		
	querytarget(j)	One of following ASCII character values (default: 2) to indicate Target: 0: A 1: B 2: AB flip (Automatically repeat inventory with another query after flipping Target flag).		
	population(y)	Integer value (as ASCII string) representing number of tags in field of view (default: 30).		
	nonexec(n)			

setantennaconfiguration(ac)
 	
(continued on next page)
	defaults(d)	None.		Command issued to set default transmit power level for all antennas to 27 dBm and use link profile 3 with tari of 0:
				`setantennaconfiguration .power 270 .linkprofileindex 3 .tari 0<CR><LF>`
				Response:
				`Command:setantennaconfiguration ,Status:OK<CR><LF>`

	power(p)	Decimal value in ASCII of output power in .1 dBm units. For example, 240 for 24 dBm. (Default: 270.)		
	linkprofileindex(lx)	Index of radio link profile to be used (default: 0).		

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters		Response	Example
	Parameter ID	Options, Default Value, Range		
setantennaconfiguration(ac) (continued from previous page)	tari(ta)	Tari value in uS to be used. For example, 6.25. Must be >= Min and <=Max tari value. If step size is supported by profile, value specified must be a multiple of step size. Defaults to first entry in link profile table. See *srfidGetSupported LinkProfiles on page 3-52* for details.		
	doselect(ds)	Select filter will be applied if it is not specified part of operation.		
	noselect(ns)	Select filter will not be applied if it is not specified part of operation.		
	noexec(n)			

setreportconfig(rc)
 	
Description: Configures fields that are reported as response to operation commands
	defaults(d)	None		Command issued to set report configuration if it is not specified part of operation.
				`setreportconfig .incphase<CR><LF>`
				Response:
				`Command: setreportconfig,Status:OK<CR><LF> <CR><LF>`

	incfirstseentime(iz)	None (default).		
	excfirstseentime(ez)	None.		
	inclastseentime(il)	None (default).		
	exclastseentime(el)	None.		
	incpc(ic)	None (default).		
	excpc(ec)	None.		
	incrssi(ir)	None (default).		
	excrssi(er)	None.		
	incphase(ik)	None.		
	excphase(ek)	None (default).		
	incchannelindex(ih)	None.		
	excchannelindex(eh)	None (default).		
	incstageencount(is)	None.		
	exctagseencount(es)	None (default).		
	noexec(n)			

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters		Response	Example
	Parameter ID	Options, Default Value, Range		

setaccesscriteria(at)
 	
Description: Sets criteria for access operation. Each criteria can have two tag filters. Up to 4 criteria can be defined and set on the reader using this command. Once added, access criteria created using this command are indexed from 1.
If one of the access criteria created using this command need to be enabled and used during an access operation, its index needs to be explicitly specified in corresponding access operation by setting useaccessfilter(uf) option with the desired access criteria index as argument.
 	
(continued on next page)
	defaults(d)	None.	One line metadata response indicate status of adding access filter terminated with <CR><LF>. As no data is associated with response, metadata line is followed by another <CR><LF> indicating end of response.	Command issued to add two access criteria:
				• First to select all tags with EPC banks having PC word as 3200 and USER bank with data as 1234 at the beginning
				• second to include all tags not having TID banks data starting with E0 and with USER bank having data 1234 at the beginning:
				`setaccesscriteria`
				`.accesscriteria`
				`.filter1maskbank epc`
				`.filter1maskstartpos 10`
				`.filter1data 3200`
				`.filter1mask FFFF`
				`.filter1matchlength 10`
				`.filter1domatch`
				`.filter2maskbank user`
				`.filter2maskstartpos 0`
				`.filter2data 1234`
				`.filter2mask FFFF`
				`.filter2matchlength 10`
				`.filter2domatch`
				Response:
				`Command:setaccesscriteria,Status:OK<CR><LF>`
				`<CR><LF>`

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters		Response	Example
	Parameter ID	Options, Default Value, Range		
setaccesscriteria(at) (continued from previous page)	accesscriteria(c)	Identifies beginning of an access criteria record. All options after this till next accesscriteria(c) option or end of command indicated by <CR><LF> (in the case of last record) compose one access criteria record.		
	filter1maskbank(q1)	One of epc/tid/user(default)/password.		
	filter1maskstartpos(a1)	4 character ASCII hex string (default: 00). Indicates start bit position from beginning of memory bank from where match pattern is checked.		
	filter1data(d1)	ASCII hex string (default: 0000). Data pattern to filter. Each character in string padded to ensure full byte.		
	filter1mask(m1)	ASCII hex string (default: 0000). Bit mask for bits to check in pattern. Each character in string padded to ensure full byte.		
	filter1matchlength(l1)	4 character ASCII hex value (default: 0010). Represents numbers of bits from start of match pattern to used for matching.		
	filter1domatch(o1)	None (default). Operate on. matching tag.		

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters		Response	Example
	Parameter ID	Options, Default Value, Range		
setaccesscriteria(at) (continued from previous page)	filter1nomatch(n1)	None. Operate on non-matching tag.		
	filter2maskbank(q2)	One of epc/tid/user (default)/password.		
	filter2maskstartpos(a2)	4 character ASCII hex string (default: 00).Indicates start bit position from beginning of memory bank from where match pattern is checked.		
	filter2data(d2)	ASCII hex string (default: 0000). Data pattern to filter. Each character in string padded to ensure full byte.		
	filter2mask(m2)	ASCII hex string (default: 0000). Bit mask for bits to check in pattern. Each character in string padded to ensure full byte.		
	filter2matchlength(l2)	4 character ASCII hex value (default: 0010). Represents numbers of bits from start of match pattern to used for matching.		
	filter2domatch(o2)	None (default). Operate on matching tag.		
	filter2nomatch(n2)	None. Operate on non-matching tag.		
	nonexec(n)			

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters		Response	Example
	Parameter ID	Options, Default Value, Range		
Operation Commands				

connect(cn)			This command to be issued before sending any ASCII command through Debug UART or USB CDC. Bluetooth will move to Connected state if no other connection present from UART or USB CDC	Command issued:
				`connect <CR><LF>`
				Response:
				`Command:connect,Status:Connection Successful<CR><LF><CR><LF>`

	override(or)	To Override the existing connection.		
	disablelowpower(dl)	To disable low power mode.		
	password(p)	To enter Bluetooth connection password.		

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters: Parameter ID	Parameters: Options, Default Value, Range	Response	Example

inventory(in)
 	
(continued on page)
	defaults(d)	None.	One line metadata of response format corresponding to fields selected by command, followed by multiple lines of response each terminated by a <CR><LF>. End of response indicated by a line with <CR><LF>.	Command issued:
				`inventory .incrssi .power 300<CR><LF>`
				Response:
				`Command:inventory Status:OK,EPCId:,Firstseentime:,Lastseentime:,PC:,RSSI:,Phase:,ChannelIndex:,TagSeenCount:`
				`,,307417001105A5866600003B,4150017718,4150017718,3000,-62,0,0,1 <CR><LF>`
				`,,AD99160040AB2D9524000030,4150033432,4150033432,3000,-59,0,0,1 <CR><LF>`
				`,,AD99160040AB1D952600002E,4150037180,4150037180,3000,-60,0,0,1 <CR><LF>`
				`,,40081234567809876543001D,4150045406,4150045406,3000,-59,0,0,1 <CR><LF>`
				`,,1111CCCCBBBBAAAA00009999,4150719034,4150719034,3000,-61,0,0,1 <CR><LF>`
				`,,8DF00000000000000812E9A,4150941225,4150941225,3000,-58,0,0,1 <CR><LF>`
				`,,443322114433221144332211,4160468335,4160468335,3000,-64,0,0,1 <CR><LF>`
				`,,AD7C090048D1158A30000011,4185639848,4185639848,3000,-62,0,0,1 <CR><LF>`
				`,,8DF0000000000000007CFEE4,4216454785,4216454785,3000,-66,0,0,1 <CR><LF>`
				`,,AD7C090048D1158A30000011,4185639848,4231389014,3000,-62,0,0,1 <CR><LF>`
				`<CR><LF>`

	batchmode(bm)	Can be one of batch mode modes: Disable: 0 Auto: 1 (default) Enable: 2.		
	incfirstseentime(iz)	None (default).		
	excfirstseentime(ez)	None.		
	inclastseentime(il)	None (default).		
	exclastseentime(el)	None.		

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters		Response	Example
	Parameter ID	Options, Default Value, Range		
inventory(in) (continued from previous page)	incpc(ic)	None (default).		
	excpc(ec)	None.		
	incrssi(ir)	None (default).		
	excrssi(er)	None.		
	incphase(ik)	None.		
	excphase(ek)	None (default).		
	incchannelindex(ih)	None.		
	excchannelindex(eh)	None (default).		
	incstageencount(is)	None.		
	exctagseencount(es)	None (default).		
	doselect(ds)	None.		
	noselect(ns)	None (default).		
	power(p)	Decimal value in ASCII of output power in .1 dBm units. For example, 240 for 24 dBm		
	setoperationconfiguration (so)			
	noexec(n)	None.		

read(rd)
 	
(continued on page)
	defaults(d)	None.	One line metadata of response format corresponding to fields selected by command, followed by multiple lines of response each terminated by a <CR><LF>.
			End of response indicated by a line with <CR><LF>.
				Command issued:
				`rd .ir .p 300<CR><LF>`
				Response:
				`Command:rd,Status:0,EPC:,USER:,RSSI:<CR><LF>`
				`,,32001122334455667788990 1,123456,-40<CR><LF>`
				`,,32001122334455667788991 2,223457,-44<CR><LF>`
				`,,32001122334455667788990 3,323454,-42<CR><LF>`
				`,,32001122334455667788990 4,423458,-40<CR><LF>`
				`<CR><LF>`
				`Note: Long format of above command will be:`
				`read .incrssi .power 300<CR><LF>`

	incfirstseentime(iz)	None (default).		
	excfirstseentime(ez)	None.		
	inclastseentime(il)	None (default).		
	exclastseentime(el)	None.		
	incpc(ic)	None (default).		
	excpc(ec)	None.		

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters		Response	Example
	Parameter ID	Options, Default Value, Range		
read(rd) (continued from previous page)	incrssi(ir)	None (default).		
	excrssi(er)	None.		
	incphase(ik)	None (default).		
	excphase(ek)	None.		
	incchannelindex(ih)	None.		
	excchannelindex(eh)	None (default).		
	doselect(ds)	None (default).		
	noselect(ns)	None.		
	power(p)	Decimal value in ASCII of output power in .1 dBm units. For example, 240 for 24 dBm		
	password(w)	8 character ASCII hex value (default:00000000) for access password		
	bank(b)	One of the following banks from where read operation needs to be performed: epc/tid/user(default)/resv		
	offset(f)	Number of words offset from beginning of data bank, from where the read operation needs to be performed. (Default: 0.)		
	length(h)	Number of words to read.		
	setoperationconfiguration(so)			
	criteriaindex(ci)			
	inctagseencount(is)			
	exctagseencount(es)			
	noexec(n)	None.		

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters			
	Parameter ID	Options, Default Value, Range	Response	Example

write(wr)
 	
(continued on next page)
	defaults(d)	None.	One line metadata of response format corresponding to fields selected by command terminated by <CR><LF>, followed by multiple lines of response each terminated by a <CR><LF>. End of response indicated by a line with <CR><LF>.	Command issued:
				`write .power 300 .password 11223344 .data ABCDEF12<CR><LF>`
				Response:
				`Command:write,Status:0,EPC:,NumWritten:,RSSI:<CR><LF>`
				`,,300012345678556677889901,4,-40<CR><LF>`
				`,,300012345678776677889912,0,-44<CR><LF>`
				`,,300012345678336677889903,4,-42<CR><LF>`
				`<CR><LF>`

	incfirstseentime(iz)	None (default).		
	excfirstseentime(ez)	None.		
	inclastseentime(il)	None (default).		
	exclastseentime(el)	None.		
	incpc(ic)	None (default).		
	excpc(ec)	None.		
	incrssi(ir)	None (default).		
	excrssi(er)	None.		
	incphase(ik)	None (default).		
	excphase(ek)	None.		
	incchannelindex(ih)	None.		
	excchannelindex(eh)	None (default).		
	doselect(ds)	None (default).		
	noselect(ns)	None.		
	power(p)	Decimal value in ASCII of output power in .1 dBm units. For example, 240 for 24 dBm		
	password(w)	8 character ASCII hex value (default:00000000) for access password		
	bank(b)	One of the following banks onto which write operation needs to be performed: epc/tid/user(default)/resv		

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters Parameter ID	Options, Default Value, Range	Response	Example
write(wr) (continued from previous page)	offset(f)	Number of words offset from beginning of data bank, from where the write operation needs to be performed. (Default: 0.)		
	data(x)	Mandatory parameter. Parameter needs to be an ASCII hex string. Each character in string padded to ensure full byte.		
	doblockwrite(br)	None (default). If parameter is present, blockwrite operation is performed with specified parameters.		
	inctagseencount(is)			
	exctagseencount(es)			
	criteriaindex(ci)			
	setoperationconfiguration (so)			
	noexec(n)	None.		

lock(lo)
 	
(continued on next page)
	defaults(d)	None.	One line metadata of response format corresponding to field's lock status by command, followed by multiple lines of response each terminated by a <CR><LF>. End of response indicated by a line with <CR><LF>.	Command issued:
				`lock .power 300 .password 11223344 .killpwd 2 .accesspwd 2 .usermem 0<CR><LF>`
				Response:
				`Command:lock,Status:0,EPC:,RSSI:<CR><LF>`
				`,,300012345678556677889901,-40<CR><LF>`
				`,,300012345678776677889912,-44<CR><LF>`
				`,,300012345678336677889903,-42<CR><LF>`
				`<CR><LF>`

	incfirstseentime(iz)	None (default).		
	excfirstseentime(ez)	None.		
	inclastseentime(il)	None (default).		
	exclastseentime(el)	None.		
	incpc(ic)	None (default).		
	excpc(ec)	None.		
	incrssi(ir)	None (default).		

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters		Response	Example
	Parameter ID	Options, Default Value, Range		
lock(lo) (continued from previous page)	excrssi(er)	None.		
	incphase(ik)	None (default).		
	excphase(ek)	None.		
	incchannelindex(ih)	None.		
	excchannelindex(eh)	None (default).		
	doselect(ds)	None (default).		
	noselect(ns)	None.		
	power(p)	Decimal value in ASCII of output power in .1 dBm units. For example, 240 for 24 dBm		
	password(w)	8 character ASCII hex value (default:00000000) for access password		
	killpwd(kp)	Lock action parameter for kill password field. If parameter is issued, one of the following options needs to be specified: 0: Writable, only from open or secure states 1: Permanently writable, only from open or secure states 2: Writable, only from secure state 3: Never writable in any states		

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters		Response	Example
	Parameter ID	Options, Default Value, Range		
lock(lo) (continued from previous page)	accesspwd(ap)	Lock action parameter for access password field. If parameter is issued,one of the following options needs to be specified: 0: Writable, only from open or secure states 1: Permanently writable, only from open or secure states 2: Writable, only from secure state 3: Never writable in any states		
	epcmem(pm)	Lock action parameter for EPC memory bank. If parameter is issued,one of the following options needs to be specified: 0: Readable and Writable, only from open or secure states. 1: Permanently Readable and Writable, only from open or secure states. 2: Readable and Writable, only from secure state. 3: Never Readable and Writable in any states.		

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters		Response	Example
	Parameter ID	Options, Default Value, Range		
lock(lo) (continued from previous page)	tidmem(tm)	Lock action parameter for TID memory bank. If parameter is issued, one of the following options needs to be specified: 0: Readable and Writable, only from open or secure states. 1: Permanently Readable and Writable, only from open or secure states. 2: Readable and Writable, only from secure state. 3: Never Readable and Writable in any states.		
	usermem(um)	Lock action parameter for User memory bank. If parameter is issued,one of the following options needs to be specified: 0: Readable and Writable, only from open or secure states. 1: Permanently Readable and Writable, only from open or secure states. 2: Readable and Writable, only from secure state. 3: Never Readable and Writable in any states.		
	inctagseencount(is)			
	exctagseencount(es)			
	criteriaindex(ci)	Index to the access criteria to be used.		
	setoperationconfiguration(so)	None.		
	noexec(n)	None.		

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters Parameter ID	Options, Default Value, Range	Response	Example

kill(kl)	defaults(d)	None.	One line metadata of response status terminated by <CR><LF>. End of response indicated by a line with <CR><LF>.	Command issued:
				`kill .power 300 .password ABCD1234<CR><LF>`
				Response:
				`Command:kill,Status:0,EPC:,RSSI:<CR><LF>`
				`,,300012345678556677889901,-40<CR><LF>`
				`,,300012345678776677889912,-44<CR><LF>`
				`,,300012345678336677889903,-42<CR><LF>`
				`<CR><LF>`

	incfirstseentime(iz)	None (default).		
	excfirstseentime(ez)	None.		
	inclastseentime(il)	None (default).		
	exclastseentime(el)	None.		
	incpc(ic)	None (default).		
	excpc(ec)	None.		
	incrssi(ir)	None (default).		
	excrssi(er)	None.		
	incphase(ik)	None (default).		
	excphase(ek)	None.		
	incchannelindex(ih)	None.		
	excchannelindex(eh)	None (default).		
	doselect(ds)	None (default).		
	noselect(ns)	None.		
	power(p)	Decimal value in ASCII of output power in .1 dBm units. For example, 240 for 24 dBm.		
	password(w)	8 character ASCII hex value (default:00000000) corresponding to kill password.		
	inctagseencount(is)			
	exctagseencount(es)			
	criteriaindex(ci)	Index to the access criteria to be used.		
	setoperationconfiguration (so)	None.		
	noexec(n)	None.		

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters		Response	Example
	Parameter ID	Options, Default Value, Range		

blockerase(be)
 	
(continued on next page)
	defaults(d)	None.	One line metadata of response status terminated by <CR><LF>. End of response indicated by a line with <CR><LF>	Command issued:
				`blockerase .password ABCD1234 .bank user .offset 4 .length 5<CR><LF>`
				Response:
				`Command:blockerase,Status:0,EPC:,RSSI:<CR><LF>`
				
				`,,300012345678556677889901,-40<CR><LF>`
				
				`,,300012345678776677889912,-44<CR><LF>`
				
				`,,300012345678336677889903,-42<CR><LF>`
				`<CR><LF>`

	incfirstseentime(iz)	None (default).		
	excfirstseentime(ez)	None.		
	inclastseentime(il)	None (default).		
	exclastseentime(el)	None.		
	incpc(ic)	None (default).		
	excpc(ec)	None.		
	incrssi(ir)	None (default).		
	excrssi(er)	None.		
	incphase(ik)	None (default).		
	excphase(ek)	None.		
	incchannelindex(ih)	None.		
	excchannelindex(eh)	None (default).		
	doselect(ds)	None (default).		
	noselect(ns)	None.		
	power(p)	Decimal value in ASCII of output power in .1 dBm units. For example, 240 for 24 dBm.		
	password(w)	8 character ASCII hex value (default:00000000) corresponding to access password		
	bank(b)	One of the following banks for which blockerases operation needs to be performed: epc/tid/user(default)/resv		

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters Parameter ID	Options, Default Value, Range	Response	Example
blockerase(be) (continued from previous page)	offset(f)	Number of words offset from beginning of data bank, from where the read operation needs to be performed. (Default: 0.)		
	length(h)	Number of words to erase.		
	setoperationconfiguration(so)			
	criteriaindex(ci)			
	inctagseencount(is)			
	exctagseencount(es)			
	setoperationconfiguration(so)	None.		
	noexec(n)	None.		

blockpermalock(bp)
 	
(continued on next page)
	defaults(d)	None.	One line metadata of response status terminated by <CR><LF>. End of response indicated by a line with <CR><LF>	Command issued:
				`blockpermalock .password ABCD1234 .dolock .blockptr 2 .blockmask f80003<CR><LF>`
				Response:
				`Command:blockpermalock,Status :0,LockStatus:<CR><LF> ,,f80003<CR><LF> <CR><LF>`

	incfirstseentime(iz)	None (default).		
	excfirstseentime(ez)	None.		
	inclastseentime(il)	None (default).		
	exclastseentime(el)	None.		
	incpc(ic)	None (default).		
	excpc(ec)	None.		
	incrssi(ir)	None (default).		
	excrssi(er)	None.		
	incphase(ik)	None (default).		
	excphase(ek)	None.		
	incchannelindex(ih)	None.		
	excchannelindex(eh)	None (default).		
	doselect(ds)	None (default).		
	noselect(ns)	None.		
	power(p)	Decimal value in ASCII of output power in .1 dBm units. For example, 240 for 24 dBm		
	dolock(pl)	None (default). If parameter is present perma lock is performed, else (default) current lok status of specified blocks are returned.		

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters		Response	Example
	Parameter ID	Options, Default Value, Range		
blockpermalock(bp) (continued from previous page)	password(w)	8 character ASCII hex value (default:00000000) corresponding to access password		
	bank(b)	One of the following banks for which blockerases operation needs to be performed: epc/tid/user (default)		
	blockptr(bt)	Starting address of blockmask in units of 16 blocks (default 0)		
	blockrange(br)	Mask range, in units of 16 blocks		
	blockmask(bm)	Bitmask representation of blocks to either perma lock (if bit asserted) or read current lock status(bit not asserted). Mandatory parameter. Parameter needs to be an ASCII hex string.		
	setoperationconfiguration(so)	None.		
	criteriaindex(ci)	Index to the access criteria to be used.		
	inctagseencount(is)			
	exctagseencount(es)			
	noexec(n)	None.		

abort(a)			One line metadata of response status terminated by <CR><LF>. End of response indicated by a line with <CR><LF>	Command issued:
				`abort <CR><LF>`
				Response:
				`Command:abort,Status:OK<CR><LF>`
				`<CR><LF>`

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters Parameter ID	Options, Default Value, Range	Response	Example

beginaccesssequence(ba)			One line metadata of response status terminated by <CR><LF>.
			End of response indicated by a line with <CR><LF>
				Command issued:
				`beginaccesssequence <CR><LF>`
				Response:
				`Command:beginaccesssequence,Status:0,SeqNum:1<CR><LF>`
				`<CR><LF>`
				Command:
				`lock power 300 .password 11223344 .data 00802<CR><LF>`
				Response:
				`Command:lock,Status:OK,SeqNum:1,CmdNum:1<CR><LF>`
				`<CR><LF>`
				Command:
				`write .data ABCDEF12<CR><LF>`
				Response:
				`Command:write,Status:OK,SeqNum:1,CmdNum:2<CR><LF>`
				`<CR><LF>`
				Command:
				`lock .power 300 .password 11223344 .data 00802<CR><LF>`
				Response:
				`Command:lock,Status:OK,SeqNum:1,CmdNum:1<CR><LF>`
				`<CR><LF>`

endactionsequence(ea)			One line metadata of response status terminated by <CR><LF>.
			End of response indicated by a line with <CR><LF>
				Command issued:
				`endactionsequence <CR><LF>`
				Response:
				`Command:endactionsequence,Status:0,SeqNum:1<CR><LF>`
				`<CR><LF>`

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters		Response	Example
	Parameter ID	Options, Default Value, Range		
execaccesssequence(xa) **Note:** Reporting and antenna config parameters specified as part of this command take precendence over correspsonding parameter specified for individual access commands that are part of this access sequence. (continued on next page)			One line metadata of response status terminated by <CR><LF>. End of response indicated by a line with <CR><LF>.	Command issued: executeactionsequence <CR><LF> Response: Command:execaccesssequence ,Status:OK,EPCId:,Firstseenti me:,RSSI:,TagSeenCount:,readS tatus:,epc:,readStatus:,tid: ,,112222223333444000000256,30 2225212,-44,1,,12753000,,E200 6004 ,,2F2203447334C3100002EB80,30 2240488,-44,1,,AFC63000,,E200 3412 ,,2F2203447334C3100002EB80,30 2256823,-44,1,,AFC63000,,E200 3412 ,,112222223333444000000256,30 2272689,-45,1,,12753000,,E200 6004 ,,2F2203447334C3100002EB80,30 2291156,-44,1,,AFC63000,,E200 3412 ,,112222223333444000000256,30 2303905,-45,1,,12753000,,E200 6004 ,,2F2203447334C3100002EB80,30 2326571,-44,1,,AFC63000,,E200 3412 ,,112222223333444000000256,30 2339173,-44,1,,12753000,,E200 6004 ,,2F2203447334C3100002EB80,30 2355104,-44,1,,AFC63000,,E200 3412 ,,112222223333444000000256,30 2368679,-45,1,,12753000,,E200 6004 ,,112222223333444000000256,30 2388045,-45,1,,12753000,,E200 6004 ,,2F2203447334C3100002EB80,30 2400630,-44,1,,AFC63000,,E200 3412 ,,112222223333444000000256,30 2417692,-44,1,,12753000,,E200 6004 ,,2F2203447334C3100002EB80,30 2431611,-44,1,,AFC63000,,E200 3412
		sequencenum(sn) (default:1).	Sequence number of action sequence previously created.	
	incfirstseentime(iz)	None (default).		
	excfirstseentime(ez)	None.		
	inclastseentime(il)	None (default).		
	exclastseentime(el)	None.		
	incpc(ic)	None (default).		

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters		Response	Example
	Parameter ID	Options, Default Value, Range		
execaccesssequence(xa) (continued from previous page)	incchannelindex(ih)	None.		
	excchannelindex(eh)	None (default).		
	doselect(ds)	None (default).		
	noselect(ns)	None.		
	power(p)	Decimal value in ASCII of output power in .1 dBm units. For example, 240 for 24 dBm		
	setoperationconfiguration(so)			
	criteriaindex(ci)			
	inctagseencount(is)			
	exctagseencount(es)			
	defaults(d)			
	noexec(n)	None. If present, access sequence is not executed. Individual access operations with parameters used for each are returned.		

locatetag(lt)	epc(ep)	EPC ID of tag to be located (max 64 chars).		Command issued:
				`lt .epc 000000000000000000002A0 .epm 000000000000000000FF0<CR><LF>`
				Response:
				`Command:locatetag ,Status:OK,Proximitypercent:<CR><LF> ,,25<CR><LF> ,,30<CR><LF>`

	epcm(epmmask)	HEX mask to be applied to the EPC ID (max 64 chars).		

gettags(tg)			One line metadata of response status terminated by <CR><LF>.
			End of response indicated by a line with <CR><LF>
				Command issued:
				`gettags<CR><LF>`
				Response:
				`Command:gettags ,Status:OK,EPCId:,Firstseentime:,RSSI:<CR><LF> ,,0000000000000000,2411552658,-38 <CR><LF> ,,000000000000000000000AD,2411547735,-43 <CR><LF> ,,00000C00B68BAE00000000B0,2411464643,-39 <CR><LF> ,,00000C00B68BDE00000000B4,2411483446,-36 <CR><LF> <CR><LF>`

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters Parameter ID	Options, Default Value, Range	Response	Example

purgetags(tp)			One line metadata of response status terminated by <CR><LF>.
			End of response indicated by a line with <CR><LF>
				Command issued:
				`purgetags <CR><LF>`
				Response:
				`Command:purgetags,Status:OK<CR><LF>`
				`<CR><LF>`

Configuration Commands				

setstarttrigger(st)
 	
Note: If startdelay is set to a non-zero value x, operation will start only after x ms since meeting start criteria except for "startonhandheldtrigger". If the start criteria is "startonhandheldtrigger" startdelay parameter is ignored for the operation start.
	startonhandheldtrigger(sp)	Enable start of operation on physical trigger pull (default).	One line metadata response indicate status of setting start trigger terminated with <CR><LF>. As no data is associated with response, metadata line is followed by another <CR><LF> indicating end of response.	Command issued to set operation start on first trigger press.
				`setstarttrigger .triggertype 0<CR><LF>`
				`Response:`
				`Command:`
				`setstarttrigger,Status:OK<CR><LF>`
				`<CR><LF>`

	ignorehandheldtrigger(ip)	Ignore state of physical trigger.		
	triggertype(tt)	Trigger type, used only if startonphysicaltrigger is set. 0: Trigger press 1: Trigger release		
	startdelay(sd)	Start operation after x ms (default: 0, immediate)		
	repeat(ro)	Repeat monitoring for start trigger after stop of operation (default).		
	dontrepeat(dr)	Complete operation after stop trigger and do not repeat monitoring for this start trigger. Set this for one-shot operation based on this trigger.		
	default(d)	None.		
	noexec(n)			

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters Parameter ID	Options, Default Value, Range	Response	Example

setstoptrigger(ot)
 	
Note: If one or more stop conditions (stoponphysicaltrigger, enablestoponcount, enablestopontimeout) are specified, operation stops when one of the conditions is encountered.
 	
(continued on next page)
	stoponhandheldtrigger(tp)	Enable stop of operation on physical trigger release (default).	One line metadata response indicate status of setting stop trigger terminated with <CR><LF>. As no data is associated with response, metadata line is followed by another <CR><LF> indicating end of response.	Command issued to set operation stop on first trigger release OR after 100 inventory rounds:
				`setstoptrigger .triggertype 1 .enablestoponinventorycount 100`
				Response command:
				`setstoptrigger,Status:OK<CR><LF>`
				`<CR><LF>`

	ignorehandheldtrigger(ip)	Ignore state of physical trigger.		
	defaults(d)	None.		
	triggertype(tt)	Trigger type, used only if startonphysicaltrigger is set. 0: Trigger press 1: Trigger release		
	enablestopontagcount(ec)	Enable stop of operation after the number of tags specified was inventoried.		
	disablestopontagcount(dc)	Disables tag count based stop (default).		
	stoptagcount(tc)	Used if enablestoponcount is set. Stop operation after n tags were inventoried since start of operation. Range for n: 0 to 65536 (default: 1).		
	enablestopontimeout(et)	Enable stop on timeout.		
	disablestopontimeout(dt)	Disables stop trigger based on timeout (default).		

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters		Response	Example
	Parameter ID	Options, Default Value, Range		
setstoptrigger(ot) (continued from previous page)	stoptimeout(to)	Used if enablestopontimeout is set. Specifies the number of milliseconds since start of operation after which the operation is stopped.		
	enablestoponinventorycount(ei)	Enable stop of operation based on number of inventory rounds completed.		
	disablestoponinventorycount(di)	Disables inventory count based stop (default).		
	stopinventorycount(si)	Used if enablestoponinventorycount is set. Stop operation after n inventory rounds since start of operation. Range for n: 0 to 65536 (default: 1).		
	enablestoponaccesscount(ea)	Enable stop of operation based on number of Access rounds completed.		
	disablestoponaccesscount(da)	Disables access count based stop (default).		
	stopaccesscont(sa)	Used if enablestoponaccesscount is set. Stop operation after n access rounds since start of operation. Range for n: 0 to 65536 (default: 1).		
	noexec(n)			

getallsupportedregions(ga)
 	
Description: Get all supported regulatory region codes.
			One line metadata of response status terminated by <CR><LF>. Followed by lines with 3 letter country code for supported regulatory regions. End of response indicated by a line with <CR><LF>	Command issued:
				`getallsupportedregions <CR><LF>`
				Response:
				`Command:getallsupportedregions,Status:OK,RegionCode:,Name: <CR><LF>`
				`,,ARG,Argentina`
				`,,AUS,"Australia"<CR><LF>`
				`,,BRZ,"Brazil"<CR><LF>`
				`,,USA,"United States of America"<CR><LF>`
				`<CR><LF>`

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters		Response	Example
	Parameter ID	Options, Default Value, Range		
getregion(gr) Description: Get current configured regulatory region.			One line metadata of response status terminated by <CR><LF>. Followed by line with 3 letter country code for current configured regulatory id or NIL if none is configured, followed by fields indicating whether frequency hopping can be configurable, followed by field indicating current status for hopping enablement, and last field either indicating enabled channels (in the case of current region) or supported channels in the case of region for which settings are explicitly queried. End of response indicated by a line with <CR><LF>	Command issued on a reader supporting FCC/worldwide SKU to get configuration for USA: `getregion .region USA<CR><LF>` Response: `Command:getregion ,Status:OK,RegionCode:,HoppingConfigurable:,SupportedChannels:<CR><LF> ,,USA,false, 915750 915250 903250 926750 926250 904250 927250 920250 919250 909250 918750 917750 905250 904750 925250 921750 914750 906750 913750 922250 911250 911750 903750 908750 905750 912250 906250 917250 914250 907250 918250 916250 910250 910750 907750 924750 909750 919750 916750 913250 923750 908250 925750 912750 924250 921250 920750 922750 902750 923250<CR><LF> <CR><LF>` Command issued on a reader supporting EU SKU to get default regulatory configuration: `getregion<CR><LF>` Response: `Command:getregion ,Status:OK,RegionCode:,HoppingConfigurable:,SupportedChannels:<CR><LF>,,GBR,true, 865700 866300 866900 867500<CR><LF> <CR><LF>`
	region(c)	Displays regulatory configurations possible for specified region. If region is not specified, display configuration for current configured region.		

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters		Response	Example
	Parameter ID	Options, Default Value, Range		

setregulatory(sg)
 	
Description: Set to specified regulatory region with region specific options.
			One line metadata of response status terminated by <CR><LF>. End of response indicated by a line with <CR><LF>.	Command issued to set device to United Kingdom regulatory region, without hopping and enable first and last channel out of the 4 supported channels in EU:
				`setregulatory .region GBR .hoppingon .enabledchannels 865700,867500 <CR><LF>`
				Response:
				`Command:setregulatory ,Status:OK<CR><LF> <CR><LF>`

	region(c)	Three letter country code. One of the following supported countries (see response of *getregion(gr) on page A-27*): • USA - United States of America • GBR - United Kingdom		
	hoppingon(ho)	Enable hopping. (default).		
	hoppingoff(hf)	Disable hopping.		
	enabledchannels(ec)	Comma separated (without spaces in between) list of channels to enable. See *getregion(gr) on page A-27* with region code option for a list of supported channels.		
	noexec(n)	Get current configured region information		

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters Parameter ID	Options, Default Value, Range	Response	Example
changeconfig(cc)			Persits configuration One line metadata of response status terminated by <CR><LF>. End of response indicated by a line with <CR><LF>.	Command issued: changeconfig <CR><LF> Response: Command:changeconfig,Status:OK<CR><LF> <CR><LF>
	mode(m)	Enter the different modes of change config.	One of the following modes are supported: saveconfig Store the current configuration to Flash savecustomdefaults Store current configuration to custom defaults restorecustomdefaults Restore from custom default area restorefactorydefaults Restore Factory defined values.	
getsupportedlinkprofiles(gp)			getsupportedlinkprofileslist One line metadata of response status terminated by <CR><LF>. End of response indicated by a line with <CR><LF>.	Command issued: getsupportedlinkprofileslist<CR><LF> Response: Command:getsupportedlinkprofileslist,Status:0,RFModeIndex:,DivideRatio:,BDR:,M:,FLM:,PIE:,MinTari:,MaxTari:,StepTari:,SpectralMaskIndicator:,EPCHAGT&CConformance<CR><LF> ,,1,64/3,640000,1,PR_ASK,1500,6250,6250,0,Dense,false<CR><LF> ,,2,64/3,640000,1,PR_ASK,2000,6250,6250,0,Dense,false<CR><LF> ,,3,64/3,120000,2,PR_ASK,1500,25000,25000,0,Dense,false<CR><LF> ,,4,64/3,120000,2,PR_ASK,1500,12500,23000,2100,Dense,false<CR><LF> ,,5,64/3,120000,2,PR_ASK,2000,25000,25000,0,Dense,false<CR><LF> <CR><LF>

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters		Response	Example
	Parameter ID	Options, Default Value, Range		
getattall(aa)			One line metadata of response status terminated by <CR><LF>. End of response indicated by a line with <CR><LF>.	Get all supported attributes.
	startattnum(a)	Integer specifying the starting attribute from which the supported attributes are to be retrieved.	One line metadata of response status terminated by <CR><LF>. End of response indicated by a line with <CR><LF>.	Get all supported attributes.
getattrinfo(ag)			One line metadata of response status terminated by <CR><LF>. End of response indicated by a line with <CR><LF>.	Get information for device attribute.
	attnum(a)	Integer specifying the attribute for which the info is requested for	One line metadata of response status terminated by <CR><LF>. End of response indicated by a line with <CR><LF>.	Get information for device attribute.
getattrinfoall(gi)			One line metadata of response status terminated by <CR><LF>. End of response indicated by a line with <CR><LF>.	Get information for all device attribute from starting attribute.
	startattnum(a)	Integer specifying the starting attribute from which the supported attributes are to be retrieved.	One line metadata of response status terminated by <CR><LF>. End of response indicated by a line with <CR><LF>.	Get information for all device attribute from starting attribute.
getnexttattrinfo(an)	attnum(a)	Integer specifying the starting attribute from which the supported attributes are to be retrieved.	One line metadata of response status terminated by <CR><LF>. End of response indicated by a line with <CR><LF>.	Get information for device attribute

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters Parameter ID	Options, Default Value, Range	Response	Example
setattr(as)			One line metadata of response status terminated by <CR><LF>. End of response indicated by a line with <CR><LF>.	Set RSM attribute.
	attnum(a)	Integer attribute number.	One line metadata of response status terminated by <CR><LF>. End of response indicated by a line with <CR><LF>.	Set RSM attribute.
	atttype(t)	Integer attribute type.	One line metadata of response status terminated by <CR><LF>. End of response indicated by a line with <CR><LF>.	Set RSM attribute.
	attvalue(u)	Integer attribute number.	One line metadata of response status terminated by <CR><LF>. End of response indicated by a line with <CR><LF>.	Set RSM attribute.
	offset(o)	Integer attribute number.	One line metadata of response status terminated by <CR><LF>. End of response indicated by a line with <CR><LF>.	Set RSM attribute.
getattoffset(ao)			One line metadata of response status terminated by <CR><LF>. End of response indicated by a line with <CR><LF>.	Get attribute value from the offset.
	attnum(a)	Integer attribute number.	One line metadata of response status terminated by <CR><LF>. End of response indicated by a line with <CR><LF>	Get attribute value from the offset.
	offset(o)			

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters		Response	Example
	Parameter ID	Options, Default Value, Range		

protocolconfig(sa)
 	
(continued on next page)
	defaults(d)	None.		Command issued:
				`protocolconfig .echoon .inccrc<CR><LF>`
				Response:
				`Command:protocolconfig ,Status:OK,CRC:50388<CR><LF> <CR><LF>`

	echoon(ee)	Echo received commands back to terminal.		
	echooff(de)	Default. Do not echo received commands from terminal.		
	debuginterface(dp)	Disable debug messages, Range 0 to 255.		
	enabledebug(ed)	Default. Enable debug messages as notifications (see notify).		
	disabledebug(dd)	Integer debug level. Possible values are: 0: Debug 1: Info 2: Warning 3: Error 4: Fatal (Default: 3.)		
	inccrc(ic)	Include checksum field for metadata line and each line in response.		
	exccrc(ec)	Default. Exclude checksum field for metadata line and each line of response.		

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters		Response	Example
	Parameter ID	Options, Default Value, Range		
protocolconfig(sa) (continued from previous page)	incoperendsummarynotify(io)	The interface where the debug message appear s. 0: Debug UART 1: Current ASCII Interface 2: As Debug Notification in Current ASCII Interface (Not Supported) 3: No Debug Message (Default: 0.)		
	excoperendsummarynotify(eo)			
	incstartoperationnotify(is)			
	excstartoperationnotify(es)			
	incstopoperationnotify(it)			
	excstopoperationnotify(et)			
	inctriggereventnotify(ig)			
	exctriggereventnotify(eg)			
	incbatteryeventnotify(ib)			
	excbatteryeventnotify(eb)			
	inctemperatureeventnotify(im)			
	exctemperatureeventnotify(em)			
	incpowereventnotify(ip)			
	excpowereventnotify(ep)			
	incdatabaseeventnotify(ia)			
	excdatabaseeventnotify(ea)			
	incradioerroreventnotify(ir)			
	excradioerroreventnotify(er)			
	incbatchmodeeventnotify(ih)			
	excbatchmodeeventnotify(eh)			
	defaults(d)			
	noexec(n)			

getversion(gv)			One line metadata of response status terminated by <CR><LF>. End of response indicated by a line with <CR><LF>.	Command issued:
				`getversion<CR><LF>`
				Response:
				`Command:getversion`
				`,Status:OK,Device:,Version:,C`
				`RC:36818<CR><LF>`
				`,,GENX_DEVICE,1.2.69,30235<CR><LF>`
				`,,BLUETOOTH,6.15,26938<CR><LF>`
				`,,NGE,1.4.44.0,41908<CR><LF>`
				`,,PL33,,4358<CR><LF>`
				`,,HARDWARE,2,46674<CR><LF>`
				`<CR><LF>`

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters		Response	Example
	Parameter ID	Options, Default Value, Range		

getcapabilities(gc)			One line metadata of response status terminated by <CR><LF>. End of response indicated by a line with <CR><LF.	Command issued:
				`getcapabilities<CR><LF>`
				
				`Response:Command:getcapabilities ,Status:OK,Name:,Value:<CR><LF> ,,SERIAL_NUMBER,720001EVT1244003<CR><LF> ,,MODEL_NAME,RFD8500-5000100-US<CR><LF> ,,MANUFACTURER_NAME,Zebra Tech Inc<CR><LF> ,,MANUFACTURING_DATE,DDMMMYY<CR><LF> ,,SACNNER_NAME,PL3307<CR><LF> ,,ASCII_VERSION,1.2.3<CR><LF> ,,SELECT_FILTERS,4<CR><LF> ,,MIN_POWER,120<CR><LF> ,,MAX_POWER,300<CR><LF> ,,POWER_STEPS,1<CR><LF> ,,AIR_PROTOCOL_VERSION,1.2.0<CR><LF> ,,MAX_ACCESS_SEQUENCE,10<CR><LF> ,,BD_ADDRESS,000000000000<CR><LF> <CR><LF>`

switchhost(sh)	snapi(s)	This command option changes interface to USB SNAPI.		Command issued:
				`switchhost .snapi (or 'switchhost .host)`
				Response:
				`Command:switchhost ,Status:OK`

	host(h)	This command option changes interface from USB to Bluetooth and vice-versa.		
	cdc(c)	This command option changes interface to USB CDC.		
	hid (i)	This command option changes interface to USB HID.		
	noexec(n)			

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters: Parameter ID	Parameters: Options, Default Value, Range	Response	Example

getdeviceinfo(gd)			Get status of different operation parameters. One line metadata of response status terminated by <CR><LF>. End of response indicated by a line with <CR><LF>.	Command issued:
				`getdeviceinfo .battery .temperature .power <CR><LF>`
				Response:
				`Notification:BatteryEvent,Cause:User Request,Level:90,Charging:true <CR><LF>`
				`Notification:TemperatureEvent,Cause:User Request,Radio AmbTemp:48,Radio PATemp:36 <CR><LF>`
				`Notification:PowerEvent,Cause:User Request,Voltage:228,Current:1,Power:5 <CR><LF>`

	battery(bt)			
	temperature(tp)			
	power(p)			

locatedevice(ld)
 	
Description:
Locates a specific RFD8500.
 	
When a specific device is located the following LED and beeper behaviors take place on the device.
• Status LED displays an amber medium flash.
• Beeper sounds five long tones, one second each.
 	
Locate Device stops when:
• it is turned off by the ZETI client
• any other ZETI command is received
• upon disconnect of the BT interface
 	
Note: Device does not enter low power mode if Locate Device is in progress. This ensures blinking of status LED.
	enable(en)	None.		Command issued:
				`ld .en<CR><LF>`
				Response:
				`Command:locatedevice,Status:OK<CR><LF>`

	disable(ds)	None.		

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters Parameter ID	Options, Default Value, Range	Response	Example

setuniquereport(ur)	enable(e)	None.		Command issued:
				`ld .en<CR><LF>`
				Response:
				`Command:locatedevice ,Status:OK<CR><LF>`

	disable(a)	None.		
	default(d)	None.		
	noexec(n)			
Notifications				
notify(nf) Description: Notifications are generated asynchronously from the device. Notification message indicates type of notification and fields reported in metadata line, followed by values in next line. Notification message is terminated by sequence of two back-to-back <CR><LF>. **Notification Types:** • Oper End • Summary • Abort • Debug Message • GPI Trigger • Battery • Temperature • Power • Start Operation • Stop Operation • Database Event • Radio Error Event • Batch Event **Note:** Notification is terminated with metadata line only and one <CR><LF>.			Notification message from sled to terminal. Will be primarily used to report alarms such as battery below critical level, temperature threshold, dropping of read data etc.	`Response: Notification: BatteryStatus,Status:OK,Level (%):95,Voltage(V):4.1, Current(mA):2200<CR><LF>`
Configuration Commands				
btpassword(btp) Description: This command changes bluetooth connection password.	password (p)	Enter new password.		Command issued: btp .p NewPassword .r NewPassword.o OldPassword
	reenter (r)	Reenter new password.		
	oldpassword (o)	Enter current password.		

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters: Parameter ID	Parameters: Options, Default Value, Range	Response	Example
Operation Commands				

authenticate(au)
 	
(continued on next page)
	defaults(d)	None.	One line metadata of response status terminated by <CR><LF>. End of response indicated by a line with <CR><LF>	Command issued:
				`authenticate .sr .il .l 80 .d 96564402375796C69664<CR><LF>`
				Response:
				`Command:authenticate,Status:0 ,EPC:,RSSI:,Response:<CR><LF> ,,E2C06F920000003A00105A43,-4 1,e920530cc781b20cfe1ab4a0144 e7335<CR><LF> <CR><LF>`

	inctimestamp(iz)	None (default).		
	exctimestamp(ez)	None.		
	incpc(ic)	None (default).		
	excpc(ec)	None.		
	incrssi(ir)	None (default).		
	excrssi(er)	None.		
	incphase(ik)	None (default).		
	excphase(ek)	None.		
	incchannelindex(ih)	None.		
	excchannelindex(eh)	None (default).		
	doselect(ds)	None (default).		
	noselect(ns)	None.		
	power(p)	Decimal value in ASCII of output power in .1 dBm units. For example, 240 for 24 dBm.		
	password(w)	8 character ASCII hex value (default:00000000) corresponding to access password.		
	sendresp(sr)	None (default). The response is sent in the response to the command.		
	storeresp(nr)	None. The response is stored in the response buffer.		
	incresplen(ip)	None (default). Include the length in the reply.		
	excresplen(el)	None. Omit the length from the reply.		

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters		Response	Example
	Parameter ID	Options, Default Value, Range		
authenticate(au) (continued from previous page)	resplen(rl)	For sent replies that do not include the length, this value must be included to indicate to the NGE the length of the reply to expect. The length is in bits.		
	csi(i)	Default: 0, ISO/IEC 29167-1, Crypto Suite Indicator. Index of the crypto suite that should be used for this authentication command.		
	msglen(l)	The length of the message in bits. The maximum values is 4095 bits.		
	msgdata(md)	This parameter is an ASCII hex string, which presents an array of 32-bit words. The message should fill the array most significate bit first. For example, for a 60-bit message, The first 32-bits should be in message[0], the next 28-bits should be the most significant bits in message[1].		
	setoperationconfiguration (so)			
	criteriaindex (ci)	1.		
	noexec(n)	None.		

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters Parameter ID	Options, Default Value, Range	Response	Example

readbuffer(rf)	defaults(d)	None.	One line metadata of response status terminated by <CR><LF>. End of response indicated by a line with <CR><LF>.	Command issued:
				`readbuffer .bc 128<CR><LF>`
				Response:
				`Command:readbuffer,Status:0,EPC:,RSSI:,Response:<CR><LF>`
				`,,E2C06F920000003A00105A43,-41,e920530cc781b20cfe1ab4a0144e7335<CR><LF>`
				`<CR><LF>`

	inctimestamp(iz)	None (default).		
	exctimestamp(ez)	None.		
	incpc(ic)	None (default).		
	excpc(ec)	None.		
	incrssi(ir)	None (default).		
	excrssi(er)	None.		
	incphase(ik)	None (default).		
	excphase(ek)	None.		
	incchannelindex(ih)	None.		
	excchannelindex(eh)	None (default).		
	doselect(ds)	None (default).		
	noselect(ns)	None.		
	power(p)	Decimal value in ASCII of output power in .1 dBm units. For example, 240 for 24 dBm		
	password(w)	8 character ASCII hex value (default:00000000) corresponding to access password		
	wordptr(wp)	Pointer to the first (16-bit) word in the readbuffer to read.		
	bitcount(bc)	The number of bits in the read buffer to read.		
	setoperationconfiguration (so)			
	criteriaindex (ci)	1.		
	noexec(n)	None.		

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters			
	Parameter ID	Options, Default Value, Range	Response	Example

untraceable(uc)
 	
(continued on next page)
	defaults(d)	None.	One line metadata of response status terminated by <CR><LF>. End of response indicated by a line with <CR><LF>	Command issued:
				`untraceable .w 12345678 .he .hu<CR><LF>`
				Response:
				`Command:untraceable,Status:0, EPC:,RSSI:<CR><LF> ,,E2C06F920000003A00105A43,-41<CR><LF> <CR><LF>`

	inctimestamp(iz)	None (default).		
	exctimestamp(ez)	None.		
	incpc(ic)	None (default).		
	excpc(ec)	None.		
	incrssi(ir)	None (default).		
	excrssi(er)	None.		
	incphase(ik)	None (default).		
	excphase(ek)	None.		
	incchannelindex(ih)	None.		
	excchannelindex(eh)	None (default).		
	doselect(ds)	None (default).		
	noselect(ns)	None.		
	power(p)	Decimal value in ASCII of output power in .1 dBm units. For example, 240 for 24 dBm		
	password(w)	8 character ASCII hex value (default:00000000) corresponding to access password		
	assertu(au)	None (default). Assert U in the XPC bits.		
	deassertu(du)	None. Deassert U in the XPC bits.		
	showepc(se)	None (default). Show EPC.		
	hideepc(he)	None, hide EPC		
	showtid(st)	None (default). Show TID.		
	hidesometid(hs)	None. Hide some TID.		
	hidealltid(ha)	None. Hide all TID.		
	showuser(su)	None (default). Show User memory.		

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters Parameter ID	Options, Default Value, Range	Response	Example
untraceable(uc) (continued from previous page)	hideuser(hu)	None. Hide user memory.		
	epclen(el)	5 LSBs: New EPC length, Values above 63 return error.		
	normalrange(nr)	None (default). Normal range.		
	togglerange(tr)	None. Toggle range temporarily.		
	reducerange(rr)	None. Reduce range.		
	setoperationconfiguration (so)			
	criteriaindex (ci)	1.		
	noexec(n)	None.		

crypto(cy)
 	
(continued on next page)
	defaults(d)	None.	One line metadata of response status terminated by <CR><LF>.
			End of response indicated by a line with <CR><LF>.
				Command issued:
				`crypto .id 1 .cl 96564402375796C69664 .ic .pf 1 .bc 1 .pm 1<CR><LF>`
				Response:
				`Command:crypto ,Status:0,EPC:,RSSI:,Response :<CR><LF>`
				
				`,,E2C06F920000003A00105A43,-41,e920530cc781b20cfe1ab4a0144e7335<CR><LF>`
				`<CR><LF>`

	inctimestamp(iz)	None (default).		
	exctimestamp(ez)	None.		
	incpc(ic)	None (default).		
	excpc(ec)	None.		
	incrssi(ir)	None (default).		
	excrssi(er)	None.		
	incphase(ik)	None (default).		
	excphase(ek)	None.		
	incchannelindex(ih)	None.		
	excchannelindex(eh)	None (default).		
	inctagseencount(is)	None.		
	exctagseencount(es)	None.		
	doselect(ds)	None (default).		
	noselect(ns)	None.		
	power(p)	Decimal value in ASCII of output power in .1 dBm units. E.g., 240 for 24 dBm.		
	password(w)	8 character ASCII hex value (default:00000000) corresponding to access password.		

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters		Response	Example
	Parameter ID	Options, Default Value, Range		
crypto(cy) (continued from previous page)	keyid(id)	The Key that should be used by the tag in its response.		
	incresplen(il)	None(default), include the length in the reply.		
	excresplen(el)	None. Omit the length from the reply.		
	challenge(cl)	This parameter is an ASCII hex string. An array of 3-32 bits words. The challenge should fill the array most significant bit first. The first 32 bits should be in IChallenge[0], the next 32 bits should be in IChallenge[1], and the final 16 bits should be the most significant bits in IChallenge[2].		
	inccustom(iu)	None. Indicates that data will be included in the response.		
	exccustom(eu)	None (default). Indicates no custom data.		
	profile(pf)	4-bit pointer that selects a memory profile for the addition of custom data. Values above 15 return an error.		
	offset(os)	Specifies a 12-bit offset (in multiples of 64-bit blocks) that needs to be added to the address that is specified by Profile. Values above 4095 return an error.		
	blockcount(bc)	4-bit number to define the size of the customer data as a number of 64-bit blocks. Values above 15 return an error.		

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters Parameter ID	Options, Default Value, Range	Response	Example
crypto(cy) (continued from previous page)	protmode(pm)	4-bit value to select the mode of operation that is used to process the custom data. Values above 15 return an error. 0: Plaintext 1: CBC (Encipherment only) 2: CMAC (Message Authentication only) 3: CBC+CMAC		
	setoperationconfiguration(so)	None.		
	criteriaindex(ci)	Index number of criteria to be used. Value between 1-32. Default: 0. Do not use access criteria.		
	noexec(n)	None.		

setdynamicpower(dp)	enable(e)	None.	One line metadata of response status terminated by <CR><LF>.
			End of response indicated by a line with <CR><LF>.
				Command issued:
				`setdynamicpower .enable<CR><LF>`
				Response:
				`Command:setdynamicpower ,Status:OK<CR><LF>`

Administrative Commands				

admconnect(acn)
 	
Description: This command should be issued after Bluetooth connection to manage SPP port for any further administrative commands to be processed.
	password(p)	Administrative password.		Command issued:
				`acn .p change<CR><LF>`
				
				Response:
				`Command:admconnect ,Status:0<CR><LF>`

admpassword(apw)
 	
Description: Change administrative connection password.
	password (p)	Enter new password.		Command issued:
				`apw .p GenX@123 .r GenX@123 .o GenXOldPassword<CR><LF>`
				
				Response:
				`Command:admpassword ,Status:0<CR><LF>`

	reenter (r)	Reenter new password.		
	oldpassword (o)	Enter current password.		
	noexec(n)			

Table A-1 *ZETI Interface Command Reference (Continued)*

Command	Parameters: Parameter ID	Parameters: Options, Default Value, Range	Response	Example

admlistconnections(alc)
 	
Description: List all active Bluetooth host connections.
				Command Issued:
				`alc<CR><LF>`
				
				Response:
				`Command:admlistconnections ,Status:0, RFID Serial Port:, SSI Scanner Serial Port: <CR><LF> ,,connected,disconnected<CR><LF>`

admdisconnect(adi)
 	
Description: Disconnect specified data connection(s).
	noexec(n)			Command issued:
				`adi .c 0<CR><LF>`
				
				Response:
				`Command:admdisconnect ,Status:0 <CR><LF>`

	connectionid(c)	Specify existing data connection to be disconnected. The values can be: 0: Disconnect all. data connections 1: Disconnect existing RFID SPP connection. 2: Disconnect existing SSI SPP connection.		

Table A-2 *Possible Select Action Values*

Action	Matching	Non-matching
0 (default)	Assert SL or inventoried → *A*	De-assert SL or inventoried → *B*
1	Assert SL or inventoried → *A*	Do nothing
2	Do nothing	De-assert SL or inventoried → *B*
3	Negate SL or (A → B, B → A)	Do nothing
4	De-assert SL or inventoried → *B*	Assert SL or inventoried → *A*
5	De-assert SL or inventoried → *B*	Do nothing
6	Do nothing	Assert SL or inventoried → *A*
7	Do nothing	Negate SL or (A → B, B → A)

Possible Errors Reported Back for ZETI Commands

Table A-3 is arranged alphabetically by command.

Table A-3 *ZETI Interface Command Errors*

Command	Parameter ID	Possible Errors

abort(a)		If there is no operation to be aborted.
		Command issued:
		`abort`
		Response:
		`Command:abort ,Status:No Radio Operation in Progress`

admconnect(acn)	password(p)	Connect failed.
		Command Issued:
		`acn .p wrongpass <CR><LF>`
		Response
		`Command:admconnect,Status:Connect failed. Wrong credentials! <CR><LF>`

admpassword(apw)	password (p)
	reenter (r)
	oldpassword (o)
		Password mismatch.
		Command Issued:
		`btp .p GenX@123 .r GenX@123 .o abcd <CR><LF>`
		Response
		`Command:admpassword ,Status:Password mismatch error <CR><LF>`

admdisconnect(adi)	connectionid(c)	Invalid connection ID.
		Command Issued:
		`adi .c 2 <CR><LF>`
		Response
		`Command:admdisconnect ,Status:Invalid connection id <CR><LF>`

authenticate(au)	password(w)	Can produce the following tag errors as per the tag's response: Tag Access unspecified error Tag Access Memory Over Run Error Tag Locked Error No Response from Tag Tag Response CRC Error Read Length Error Access Criteria Not Matching Tag Password Error Tag Access Barker Error Tag Access Length Bit Parity Error Tag Access Regulatory Timeout Error Tag Access OLIO Timeout Error Tag Access Radio Dwell Timeout Error Tag Access IO Stop Error Tag Access Stop Request Error Tag Access Cause Unknown Error Radio Operation Start Error
	criteriaindex(ci)	Range Validation.
	msglen(l)	Range Validation and Authenticate Message Mismatch.
	csi(i)	Range Validation.
	resplength(rp)	Range Validation.
	msgdata(md)	Range Validation.
	power(p)	Range Validation.

Table A-3 *ZETI Interface Command Errors (Continued)*

Command	Parameter ID	Possible Errors

beginaccesssequence(ba)		If the number of access operation added are more than the limit(10):
		Max limit reached for Access Sequence
		noexe is not allowed in access seq:
		Command issued:
		`ba`
		Response
		`Command:beginaccesssequence ,Status:OK`
		Command issued
		`rd .n`
		Response:
		`Command:read ,Status:noexec option during Access Sequence`

blockerase(be)	defaults(d)	Can produce following tag errors as per the tag's response: Tag Access unspecified error Tag Access Memory Over Run Error Tag Locked Error No Response from Tag Tag Response CRC Error Read Length Error Access Criteria Not Matching Tag Password Error Tag Access Barker Error Tag Access Length Bit Parity Error Tag Access Regulatory Timeout Error Tag Access OLIO Timeout Error Tag Access Radio Dwell Timeout Error Tag Access IO Stop Error Tag Access Stop Request Error Tag Access Cause Unknown Error Radio Operation Start Error
	power(p)	Range Validation.
	password(w)	Range Validation.
	bank(b)	Range Validation.
	offset(f)	Range Validation.
	length(h)	Range Validation.
	criteriaindex(ci)	Range Validation.

Table A-3 *ZETI Interface Command Errors (Continued)*

Command	Parameter ID	Possible Errors
blockpermalock(bp)	defaults(d)	Can produce following tag errors as per the tag's response: Tag Access unspecified error Tag Access Memory Over Run Error Tag Locked Error No Response from Tag Tag Response CRC Error Read Length Error Access Criteria Not Matching Tag Password Error Tag Access Barker Error Tag Access Length Bit Parity Error Tag Access Regulatory Timeout Error Tag Access OLIO Timeout Error Tag Access Radio Dwell Timeout Error Tag Access IO Stop Error Tag Access Stop Request Error Tag Access Cause Unknown Error Radio Operation Start Error
	power(p)	Range Validation.
	password(w)	Range Validation.
	bank(b)	Range Validation.
	blockptr(bt)	Range Validation.
	blockrange(br)	Range Validation.
	blockmask(bm)	Range Validation.
	criteriaindex(ci)	Range Validation.
changeconfig(cc)	mode(m)	In case the save fails on the flash due to any reason: Save Config failed
connect(cn)	password(p)	Password mismatch error.

execaccesssequence(xa)	incfirstseentime(iz)	If the Access Sequence is empty:
		Command issued:
		`xa`
		Response:
		`Command:execaccesssequence ,Status:Empty Access Sequence`

	power(p)	Range Validation.
	criteriaindex(ci)	Range Validation.

getdeviceinfo(gd)	battery(bt)	For Incoorect option:
		Command issued:
		`gd .abcd`
		Response
		`Command:getdeviceinfo ,Status:abcd - Command Option not found`

getregion(gr)	region(c)	For unsupported Regions (e.g., issued in FCC SKU)
		Command issued:
		`gr .c IND`
		Response
		`Command:getregion ,Status:Region Support not Present`

Table A-3 *ZETI Interface Command Errors (Continued)*

Command	Parameter ID	Possible Errors

gettags(tg)		If attempting gettags in batch mode, disable during inventory opr
		Command issued:
		`tg`
		Response:
		`Command:gettags ,Status:Running Condition Mismatch-Command Not Allowed`

inventory(in)	defaults(d)	In case the radio doesn't start because of any error: Radio Operation Start Error
	batchmode(bm)	Range Validation.
	power(p)	Range Validation.
kill(kl)	defaults(d)	Can produce following tag errors as per the tag's response: Tag Access unspecified error Tag Access Memory Over Run Error Tag Locked Error No Response from Tag Tag Response CRC Error Read Length Error Access Criteria Not Matching Tag Password Error Tag Access Barker Error Tag Access Length Bit Parity Error Tag Access Regulatory Timeout Error Tag Access OLIO Timeout Error Tag Access Radio Dwell Timeout Error Tag Access IO Stop Error Tag Access Stop Request Error Tag Access Cause Unknown Error Radio Operation Start Error
	power(p)	Range Validation.
	password(w)	Range Validation.
	criteriaindex(ci)	Range Validation.
locatetag(lt)	epc(ep)	Range Validation.

Table A-3 *ZETI Interface Command Errors (Continued)*

Command	Parameter ID	Possible Errors
lock(lo)	defaults(d)	Can produce following tag errors as per the tag's response: Tag Access unspecified error Tag Access Memory Over Run Error Tag Locked Error No Response from Tag Tag Response CRC Error Read Length Error Access Criteria Not Matching Tag Password Error Tag Access Barker Error Tag Access Length Bit Parity Error Tag Access Regulatory Timeout Error Tag Access OLIO Timeout Error Tag Access Radio Dwell Timeout Error Tag Access IO Stop Error Tag Access Stop Request Error Tag Access Cause Unknown Error Radio Operation Start Error
	power(p)	Range Validation.
	password(w)	Range Validation.
	killpwd(kp)	Range Validation.
	accesspwd(ap)	Range Validation.
	epcmem(pm)	Range Validation.
	tidmem(tm)	Range Validation.
	usermem(um)	Range Validation.
	criteriaindex(ci)	Range Validation.
protocolconfig(sa)	debuginterface(dp)	Range Validation.

Table A-3 *ZETI Interface Command Errors (Continued)*

Command	Parameter ID	Possible Errors

read(rd)	defaults(d)	Can produce following tag errors as per the tag's response:
		Tag Access unspecified error
		Tag Access Memory Over Run Error
		Tag Locked Error
		No Response from Tag
		Tag Response CRC Error
		Read Length Error
		Access Criteria Not Matching
		Tag Password Error
		Tag Access Barker Error
		Tag Access Length Bit Parity Error
		Tag Access Regulatory Timeout Error
		Tag Access OLIO Timeout Error
		Tag Access Radio Dwell Timeout Error
		Tag Access IO Stop Error
		Tag Access Stop Request Error
		Tag Access Cause Unknown Error
		Radio Operation Start Error
		
		For example:
		Command issued:
		`rd .b user .h 2`
		Response:
		`Command:read`
		`,Status:OK,EPCId:,Firstseentime:,RSSI:,TagSeenCount:,readStatus:,user:`
		`,,0777,2095096735,-64,1,Tag Access Memory Over Run Error`
		`,,11220C00B68BB800000000B1,2095105154,-68,1,,00000000`
		`,,8DF000000000000000812447,2095116685,-68,1,Tag Access Memory Over Run Error`
		`,,11220C00B68BD000000000B3,2095128243,-67,1,,00000000`
		`,,E2002849491500901000B0D2,2095139479,-76,1,Tag Response CRC Error`

	power(p)	Range Validation.
	bank(b)	Range Validation.
	offset(f)	Range Validation.
	length(h)	Range Validation.
	criteriaindex(ci)	Range Validation.

Table A-3 *ZETI Interface Command Errors (Continued)*

Command	Parameter ID	Possible Errors
readbuffer(rf)	password(w)	Can produce following tag errors as per the tag's response: Tag Access unspecified error Tag Access Memory Over Run Error Tag Locked Error No Response from Tag Tag Response CRC Error Read Length Error Access Criteria Not Matching Tag Password Error Tag Access Barker Error Tag Access Length Bit Parity Error Tag Access Regulatory Timeout Error Tag Access OLIO Timeout Error Tag Access Radio Dwell Timeout Error Tag Access IO Stop Error Tag Access Stop Request Error Tag Access Cause Unknown Error Radio Operation Start Error
	criteriaindex(ci)	Range Validation.
	wordptr(wp)	Range Validation.
	bitcount(bc)	Range Validation.
	power(p)	Range Validation.

Table A-3 *ZETI Interface Command Errors (Continued)*

Command	Parameter ID	Possible Errors

setaccesscriteria(at)	defaults(d)	Multiple access filters are not allowed.
		Command issued:
		`setaccesscriteria .c .q1 epc .a1 2 .l1 1 .d1 E200 .m1 FFFF .o1 .c .q1 epc`
		Response:
		`Command:setaccesscriteria ,Status: Only One Access Criteria is Allowed`
		Setting Filter 2 without setting filter 1
		Command issued:
		`at .c .q2 epc .a2 2 .l2 1 .d2 2f22 .m2 2f22 .o2`
		Response:
		`Command:setaccesscriteria ,Status:Setting Filter2 Without Setting Filter1 is not Allowed`
		when the match length is not equal to the match pattern's length:
		Command issued:
		`at .c .q1 epc .a1 2 .l1 4 .d1 1122 .m1 ffff .o1`
		Response:
		`Command:setaccesscriteria ,Status:Invalid Filter1 Settings Not Allowed`

	filter1maskbank(q1)	Range Validation.
	filter1maskstartpos(a1)	Range Validation.
	filter1data(d1)	Range Validation.
	filter1mask(m1)	Range Validation.
	filter1matchlength(l1)	Range Validation.
	filter2maskbank(q2)	Range Validation.
	filter2maskstartpos(a2)	Range Validation.
	filter2data(d2)	Range Validation.
	filter2mask(m2)	Range Validation.
	filter2matchlength(l2)	Range Validation.

Table A-3 *ZETI Interface Command Errors (Continued)*

Command	Parameter ID	Possible Errors

setantennaconfiguration(ac)	defaults(d)	Depending upon the Link Profile selected the following TARI validation error are possible.
		Command issued:
		`setantennaconfiguration .noexec`
		Response:
		`Command:setantennaconfiguration .power 270 .linkprofileindex 0 .tari 0 .noselect .noexec:1,Status:OK`
		Command issued:
		`setantennaconfiguration .tari 30000`
		Response:
		`Command:setantennaconfiguration ,Status:Tari Value is Not Valid`
		Also, depending on the supported Link Profile the following is possible:
		Command issued:
		`ac .lx 2`
		Response:
		`Command:setantennaconfiguration ,Status:Link Profile Index is Not Supported`

	power(p)	Range Validation.
	linkprofileindex(lx)	Range Validation.
	tari(ta)	Range Validation.
setqueryparams(qp)	queryselect(e)	Range Validation.
	querysession(i)	Range Validation.
	querytarget(j)	Range Validation.
	population(y)	Range Validation.

setregulatory(sg)	region(c)	For unsupported frequency.
		Command issued:
		`setregulatory .region GBR .hoppingon .enabledchannels 867501`
		Response:
		`Command:setregulatory ,Status:Freqency Not found in the specified Region`
		For unsupported Regions(eg. issued in EU SKU)
		Command issued:
		`setregulatory .region USA`
		Response:
		`Command:setregulatory ,Status:Region Support not Present`
		While not enabling any channel:
		Command issued:
		`setregulatory .region IND .ec`
		Response:
		`Command:setregulatory ,Status:Atleast one Channel should be Enabled`

Table A-3 *ZETI Interface Command Errors (Continued)*

Command	Parameter ID	Possible Errors
setselectrecords(sr)	target(g)	Range Validation.
	action(o)	Range Validation.
	maskbank(q)	Range Validation.
	maskstartpos(a)	Range Validation.
	matchpattern(m)	Range Validation.
	matchlength(l)	Range Validation.
setstarttrigger(st)	triggertype(tt)	Range Validation.
	startdelay(sd)	Range Validation.
setstoptrigger(ot)	triggertype(tt)	Range Validation.
	stoptagcount(tc)	Range Validation.
	stoptimeout(to)	Range Validation.
	stopinventorycount(si)	Range Validation.
	stopaccesscont(sa)	Range Validation.
untraceable(uc)	password(w)	Can produce following tag errors as per the tag's response: Tag Access unspecified error Tag Access Memory Over Run Error Tag Locked Error No Response from Tag Tag Response CRC Error Read Length Error Access Criteria Not Matching Tag Password Error Tag Access Barker Error Tag Access Length Bit Parity Error Tag Access Regulatory Timeout Error Tag Access OLIO Timeout Error Tag Access Radio Dwell Timeout Error Tag Access IO Stop Error Tag Access Stop Request Error Tag Access Cause Unknown Error Radio Operation Start Error
	criteriaindex(ci)	Range Validation.
	epclen(el)	Range Validation.
	power(p)	Range Validation.

Table A-3 *ZETI Interface Command Errors (Continued)*

Command	Parameter ID	Possible Errors
write(wr)	defaults(d)	Can produce following tag errors as per the tag's response: Tag Access unspecified error Tag Access Memory Over Run Error Tag Locked Error No Response from Tag Tag Response CRC Error Read Length Error Access Criteria Not Matching Tag Password Error Tag Access Barker Error Tag Access Length Bit Parity Error Tag Access Regulatory Timeout Error Tag Access OLIO Timeout Error Tag Access Radio Dwell Timeout Error Tag Access IO Stop Error Tag Access Stop Request Error Tag Access Cause Unknown Error Radio Operation Start Error
	power(p)	Range Validation.
	password(w)	Range Validation.
	bank(b)	Range Validation.
	offset(f)	Range Validation.
	data(x)	Field can only take word values and Range Validation.
	criteriaindex(ci)	Range Validation.

Generic Errors Applicable to ZETI Commands

Table A-4 *Generic Error Messages*

Error Messages
Operation in progress-command not allowed
Memory allocation failed
Bluetooth error
Radio operation start error
Charging in progress-command not allowed
Operation in progress-command not allowed
Command not supported
Command option parse error
Option not allowed for this command
Command option not found
Command option type not found
Mandatory parameter missing
Command option without delimiter
Response type not found
Metadata process error
No execute process error
Not in test mode. command not allowed
Unknown GPIO port
Value out of range
Value not valid
Value not allowed
Value not present
ASCII connection not present
ASCII connection already exists
Command not allowed- region not set
Invalid char present in the value
Value string limit exceeded
Max allowed size exceeded
Size less than allowed
Field can only take word values

Radio Protocol Specific Errors Returned For An Operation

Table A-5 *Radio Protocol Error Messages*

Error Messages
Radio response timeout
Tag access unspecified error
Tag access memory over run error
Tag locked error
Tag insufficient power
No response from tag
Tag response CRC error
Read length error
Access criteria not matching
Tag password error
Tag access barker error
Tag access length bit parity error
Tag access regulatory timeout error
Tag access olio timeout error
Tag access radio dwell timeout error
Tag access IO stop error
Tag access stop request error
Tag access cause unknown error
Single channel is allowed in non-hopping
Frequency not found in the specified region
Hopping configuration not supported
Only one access criteria is allowed
Hopping is not allowed for this region

Command Specific Errors for Different ZETI Commands

Table A-6 *Command Error Messages*

Error Messages
Region support not present
At least one channel should be enabled
Save config failed
Link profile index is not supported
Tari value is not valid
Only one access criteria is allowed
Hopping is not allowed for this region
Password mismatch error
Invalid filter1 settings not allowed
Setting filter2 without setting filter1 is not allowed
Max limit reached for access sequence
No exec option during access sequence
Empty access sequence
Access sequence save operation not supported
Data field cannot be empty for write
No operation in progress
Authenticate message mismatch
Running condition mismatch-command not allowed
Save operation not allowed in test mode
Pass through already in progress
Not in pass-through mode

Appendix B COMMANDS and ATTRIBUTE REFERENCES

Table B-1 *Commands Saved on Non Volatile Medium Automatically*

Command	Description
setregulatory	Sets regulatory information.

Table B-2 *Attributes Set by the setattrib Command*

Attribute	Attribute Number	Type	Value	Description
Beeper Volume	140	Byte	0-2	0 = High 1 = Medium 2 = Low
BT Mode	383	Byte	15, 17, 19	22 = SPP and Mfi combination 17 = HID Keyboard Emulation

Table B-3 *Commands*

Command(short)	Description
connect(cn)	Command to establish ASCII connection.
abort(a)	Command to abort the Current Operation.
authenticate(au)	Command to Authenticate.
beginaccesssequence(ba)	Begin Access Sequence.
blockerase(be)	Command to Erase Block of Memory.
blockpermalock(bp)	Command to Lock Block of Memory.
changeconfig(cc)	Save Current Config as default.

Table B-3 *Commands (Continued)*

Command(short)	Description
crypto(cy)	Command to Crypto.
endaccesssequence(ea)	End Access Sequence.
execaccesssequence(xa)	Execute Access Sequence.
getallsupportedregions(ga)	Get the Supported Regions.
getattall(aa)	Get all supported attributes.
getattoffset(ao)	Get attribute value from the offset.
getattrinfo(ag)	Get information for device attribute.
getattrinfoall(gi)	Get information for all device attribute from starting attribute.
getcapabilities(gc)	Get Different Capabilities.
getdeviceinfo(gd)	Get Device Status.
getnexttattrinfo(an)	Get information of next valid attribute.
getregion(gr)	Get Configuration of Region.
getsupportedlinkprofiles(gp)	Get the Supported Link Profiles.
gettags(tg)	Get the batch mode Tags.
getversion(gv)	Get Different Versions.
inventory(in)	ommand to invoke the Inventory.
kill(kl)	Command to Kill RFID Tag.
locatetag(lt)	Locate Tag Specified.
lock(lo)	Command to Lock Memory Field.
protocolconfig(sa)	Set ASCII Configuration.
purgetags(tp)	Purge batch mode Tags.
read(rd)	Command to Read Memory Bank.
readbarcode(rb)	Read bar code.
readbuffer(rf)	Command to Read Buffer.
setqueryparams(qp)	Command to Set Query Parameters.
setaccesscriteria(at)	Set Access Criteria.
setantennaconfiguration(ac)	Command Set Antenna Config.
setattr(as)	set RSM attribute.
setdynamicpower(dp)	Command to Set Dynamic Power.
setdutyCycle(dc)	Command to Set Duty Cycle.
setregulatory(sg)	Set Regulatory for RFID.

Table B-3 *Commands (Continued)*

Command(short)	Description
setreportconfig(rc)	Command to set Report config for Tag Report.
setselectrecords(sr)	Command Set prefilters.
setstarttrigger(st)	Set Start Trigger Configuration.
setstoptrigger(ot)	Set Stop Trigger Configuration.
untraceable(uc)	Command to Untraceable.
write(wr)	Command to Write Memory Bank.
btpassword(btp)	Set Bluetooth password.

Appendix C NFC BASED CONNECTION to RFD8500i USING the ANDROID APPLICATION DEVELOPER

General Application NFC Implementation for RFD8500i

Refer to the Android documentation on NFC to get overall NFC design and usage information for an Android device. Use the following link to access the Zebra Developer Support portal for EMDK documentation and information about the EMDK functionality for NFC in Android:
http://developer.android.com/guide/topics/connectivity/nfc/nfc.html

RFD8500i NFC tags are programmed with `application/zeb.bluetooth.ep.oob` custom mimetype and are fully Bluetooth format compliant NFC forum type 2 (includes BT device address and friendly name).

The application can define the intent filter in the launcher activity to automatically launch whenever the NFC tag is read by the RFD8500i.

```
<intent-filter>
    <action android:name="android.nfc.action.NDEF_DISCOVERED"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:mimeType="application/zeb.bluetooth.ep.oob"/>
</intent-filter>
<intent-filter>
    <action android:name="android.nfc.action.TAG_DISCOVERED"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:mimeType="application/zeb.bluetooth.ep.oob"/>
</intent-filter>
```

Depending on the application it may not be necessary to launch automatically on NFC tag detection until the user launches the application. In this case use the following code in the intent filter.

```
<category android:name="android.intent.category.INFO"/>
```

If the application prevents the application launch it is necessary to disable and enable NFC when the application is resumed.

EMDK Usage for NFC Functionality

This section describes the usage of the Zebra EMDK and provides code snippets for achieving NFC functionality on Zebra Enterprise devices.

The code snippets in the sections below are extracted from the Android Zebra RFID Mobile application which demonstrates the usage of the NFC and RFID functionality in Android devices. Refer to the Android Zebra RFID Mobile application for more information about the Java code.

Initialization

As part of the initialization the EMDK object creation must be coded for NFC handling. Useful code snippets are shown below.

```
//Declare a variable to store ProfileManager object
private ProfileManager vProfileManager = null;
//Declare a variable to store EMDKManager object
private EMDKManager vEmdkManager = null;
public boolean EMDKinstalled = false;
try {
        Class.forName( "com.symbol.emdk.EMDKManager" );
        EMDKinstalled = true;
} catch( final ClassNotFoundException e ) {
        //Toast.makeText(activityObject.getApplicationContext(), "EMDK is not available",
Toast.LENGTH_SHORT).show();
        return;
}
try {
        EMDKManager.EMDKListener el = new EMDKManager.EMDKListener() {
            @Override
            public void onOpened(EMDKManager emdkManager) {
                //This callback will be issued when the EMDK is ready to use.
                vEmdkManager = emdkManager;

                bcConnection = new Bc();
                bcConnection.init(vEmdkManager);

                //Get the ProfileManager object to process the profiles
                vProfileManager = (ProfileManager)
emdkManager.getInstance(EMDKManager.FEATURE_TYPE.PROFILE);
                if(Application.disableNFC){
                    ((BaseReceiverActivity)activityObject).disableNFC();
                    //Toast.makeText(activityObject.getApplicationContext(), "Connecting ...",
Toast.LENGTH_LONG).show();
                    ((BaseReceiverActivity) activityObject).enableNFC();
                }

            }

            @Override
            public void onClosed() {
                if (vEmdkManager != null) {
                    vEmdkManager.release();
                    vEmdkManager = null;
                }
            }
        };
```

(continued on next page)

```
        //The EMDKManager object will be created and returned in the callback.
        EMDKResults results = EMDKManager.getEMDKManager(activityObject.getApplicationContext(), el);
        if (!nfcConnection.mNfcAdapter.isEnabled()) {
            if (results.statusCode == EMDKResults.STATUS_CODE.SUCCESS) {
                //EMDKManager object creation success
                Toast.makeText(activityObject.getApplicationContext(), "Enabling NFC ...",
Toast.LENGTH_LONG).show();
            } else {
                Toast.makeText(activityObject.getApplicationContext(),
Defines.NFC_CONNECT_MESSAGE_FAILED, Toast.LENGTH_LONG).show();
            }
        }

} catch (Exception ex) {
        // Toast.makeText(activityObject.getApplicationContext(), Defines.NFC_CONNECT_MESSAGE_FAILED,
Toast.LENGTH_LONG).show();
}
```

Connecting on Receiving the NFC Intent

```
public void onNewIntent(Intent intent) {
        try {
        //Get the NFC data and use the BT address to pair and connect.
                nfcInfoData = nfcConnection.getNfcData();
                if (nfcInfoData.size() > 0) {
                    for (int c1 = 0; c1 < nfcInfoData.size(); c1++) {
                        if(nfcInfoData.get(c1) != null ){
                            recievedNdefRecord=getNdefRecord(nfcInfoData.get(c1));
                         recvdMacAddress = recievedNdefRecord.deviceAddress.replaceAll("(.{2})(?!$)",
"$1:");
                            if(btConnection.isValidMacAddress(recvdMacAddress))
               // Pair if necessary and Connect to the RFD8500i device
                                //connecting_pairingFlag = pairConnect(recvdMacAddress, true);
                            else{
                                //showToast(recvdMacAddress + " is not valid BT address");
                            }
                        }
                    }
                }
            }
        }
        catch(Exception ex) {
            AllowNfcHandling();
            ////showToast("EXCEPTION(NfcBT) - 'onNewIntent'");
        }
    }
```

(continued on next page)

```
//#####################################
NdefRecord getNdefRecord(String nfcData){
        NdefRecord ndefRecord = null;
        if(nfcData != null){
            ndefRecord = new NdefRecord();
            int ndefRecordLength =
Integer.parseInt(ByteArrayUtil.byteArrayToHexString(ByteArrayUtil.reverseByteArray(ByteArrayUtil.hexS
tringToByteArray(nfcData.substring(0, 4)))), 16);
            ndefRecord.deviceAddress =
ByteArrayUtil.byteArrayToHexString(ByteArrayUtil.reverseByteArray(ByteArrayUtil.hexStringToByteArray(
nfcData.substring(4, 16))));
            nfcData = nfcData.substring(16);
            while(!nfcData.isEmpty()){
                int dataLenth=Integer.parseInt(nfcData.substring(0,2),16);
                String dataType=nfcData.substring(2,4);
                String data=  new
String(ByteArrayUtil.hexStringToByteArray(nfcData.substring(4,(dataLenth-1)*2+4)));
                switch (dataType) {
                    case Defines.CLASS_OF_DEVICE:
                                                        ndefRecord.classOfDevice = data;
                                                        break;
                    case Defines.UUID:
                                                        ndefRecord.UUIDList = data;
                                                        break;
                    case Defines.BT_LOCAL_NAME:
                                                        ndefRecord.btLocalName = data;
                                                        break;
                }
                nfcData = nfcData.substring((dataLenth+1)*2);
            }
        }
        return ndefRecord;
}
```

Setting up Foreground Dispatchers

The application can setup foreground dispatchers to capture all NFC tag read events whenever the application is in foreground.

```
    /**
     * @param activity The corresponding {@link Activity} requesting the foreground dispatch.
     * @param adapter The {@link android.nfc.NfcAdapter} used for the foreground dispatch.
     */
    public static void setupForegroundDispatch(final Activity activity, NfcAdapter adapter) {
        final Intent intent = new Intent(activity, activity.getClass());
        intent.setFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP);

        final PendingIntent pendingIntent = PendingIntent.getActivity(activity, 0, intent, 0);

        IntentFilter[] filters = new IntentFilter[3];
        String[][] techList = new String[][]{};

        // Notice that this is the same filter as in our manifest.
        filters[0] = new IntentFilter();
        filters[0].addAction(NfcAdapter.ACTION_NDEF_DISCOVERED);
        filters[0].addCategory(Intent.CATEGORY_DEFAULT);
        filters[1] = new IntentFilter();
        filters[1].addAction(NfcAdapter.ACTION_TAG_DISCOVERED);
        filters[1].addCategory(Intent.CATEGORY_DEFAULT);
        filters[2] = new IntentFilter();
        filters[2].addAction(NfcAdapter.ACTION_TECH_DISCOVERED);
        filters[2].addCategory(Intent.CATEGORY_DEFAULT);
        try {
            filters[0].addDataType(MIME_TEXT_PLAIN);
        } catch (IntentFilter.MalformedMimeTypeException e) {
            throw new RuntimeException("Check your mime type.");
        }
        adapter.enableForegroundDispatch(activity, pendingIntent, filters, techList);
    }
```

Disabling/Enabling NFC

The below code demonstrates usage of the EMDK for disabling and enabling NFC.

```
//##############################################
    private void changeNFCSettings(int value) {
        String[] modifyData = new String[1];
        modifyData[0] =
                "<?xml version=\"1.0\" encoding=\"utf-8\"?>" +
                        "<characteristic type=\"Profile\">" +
                        "<parm name=\"ProfileName\" value=\"" + profileName + "\"/>" +
                        "<characteristic type=\"WirelessMgr\">" +
                        "<parm name=\"NFCState\" value=\"" + value + "\"/>" +
                        "</characteristic>" +
                        "</characteristic>";

        new ProcessProfileTask().execute(modifyData[0]);
    }
//##############################################
    private class ProcessProfileTask extends AsyncTask<String, Void, EMDKResults> {

        @Override
        protected EMDKResults doInBackground(String... params) {

            //Call process profile to modify the profile of specified profile name
            EMDKResults results = vProfileManager.processProfile(profileName,
ProfileManager.PROFILE_FLAG.SET, params);
```

(continued on next page)

```
return results;
        }

        @Override
        protected void onPostExecute(EMDKResults results) {

            super.onPostExecute(results);

            //Check the return status of processProfile
            if(results.statusCode == EMDKResults.STATUS_CODE.CHECK_XML) {

                // Get XML response as a String
                String statusXMLResponse = results.getStatusString();

                try {
                    // Create instance of XML Pull Parser to parse the response
                    XmlPullParser parser = Xml.newPullParser();
                    // Provide the string response to the String Reader that reads
                    // for the parser
                    parser.setInput(new StringReader(statusXMLResponse));
                    // Call method to parse the response
                    parseXML(parser);
                    if ( TextUtils.isEmpty(parmName) && TextUtils.isEmpty(errorType) &&
TextUtils.isEmpty(errorDescription) ) {
                        if(!TextUtils.isEmpty(NFCState)){
                            if(NFCState.equals("1")) {
                                if(!Application.disableNFC) {
                                    Toast.makeText(activityObject.getApplicationContext(),
Defines.NFC_CONNECT_MESSAGE_ENABLED, Toast.LENGTH_LONG).show();
                                }else{
                                    Application.disableNFC=false;
                                }
                                Application.NFC = true;
                                if (nfcConnection.mNfcAdapter != null) {
                                    if(BluetoothAdapter.getDefaultAdapter().isEnabled())

setupForegroundDispatch(activityObject,nfcConnection.mNfcAdapter);
                                     //nfcConnection.mNfcAdapter.enableForegroundDispatch(activityObject,
nfcConnection.mPendingIntent, null, null);
                                }
                            }
                            else if(NFCState.equals("2"))
                                if(!Application.disableNFC) {
                                    Toast.makeText(activityObject.getApplicationContext(),
Defines.NFC_CONNECT_MESSAGE_DISABLED, Toast.LENGTH_LONG).show();
                                }

                        }
                    }
                    else {
                        Toast.makeText(activityObject.getApplicationContext(),
Defines.NFC_CONNECT_MESSAGE_FAILED, Toast.LENGTH_LONG).show();
                    }
                } catch (XmlPullParserException e) {
                    Toast.makeText(activityObject.getApplicationContext(),
Defines.NFC_CONNECT_MESSAGE_FAILED, Toast.LENGTH_LONG).show();
                }
            }
        }
    }
```

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

MN002222A03 Revision A - September 2016