RFD8500 RFID DEVELOPER GUIDE

RFD8500 RFID DEVELOPER GUIDE

MN002222A02

Revision A

March 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

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 based devices.

Chapter Descriptions

This guide includes the following topics:

Related Documents

- Zebra Scanner SDK for Android Developer Guide, p/n MN002223Axx.
- Zebra Scanner SDK for iOS Developer Guide, p/n MN001834Axx.
- RFD8500 User Guide, p/n MN002065Axx.
- RFD8500 Quick Start Guide, p/n MN002225Axx.
- RFD8500 Regulatory Guide, p/n MN002062Axx.
- CRDUNIV-RFD8500-1R Three Slot Universal Charge Only Cradle Regulatory Guide, p/n MN002224Axx.

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 Mobile 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.

✓ ***NOTE*** Some testing was performed with MC40 4.1.1, and TC55 v4.1.2.

- 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 5 Touch, 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 DUT, 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 Mobile 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.
- 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 the Zebra Scanner Control for Android application are provided using the SSI protocol in BT SPP mode. See *Related Documents on page xii* for information on the scanner developer guides.

Pairing with Bluetooth

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

- BT friendly name is printed on the device sticker on the back of the antenna.
- Device S/N is printed below 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 an Android Phone

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.

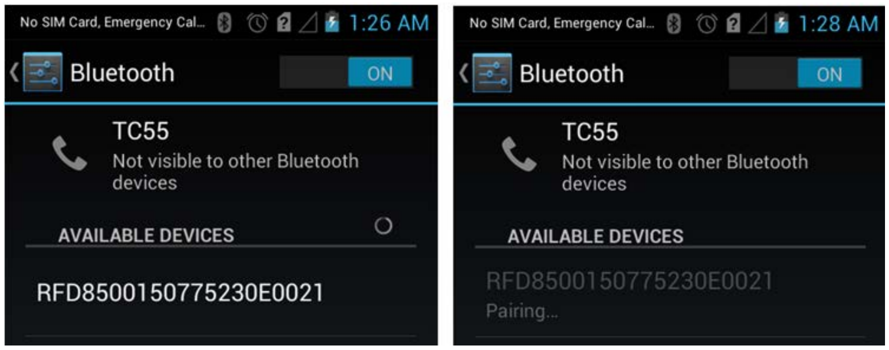

No SIM Card, Emergency Cal...
1:26 AM
Bluetooth
ON
TC55
Not visible to other Bluetooth devices
AVAILABLE DEVICES
RFD8500150775230E0021
No SIM Card, Emergency Cal...
1:28 AM
Bluetooth
ON
TC55
Not visible to other Bluetooth devices
AVAILABLE DEVICES
RFD8500150775230E0021
Pairing...

Figure 1-1 *Pairing with an Android Phone*

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.

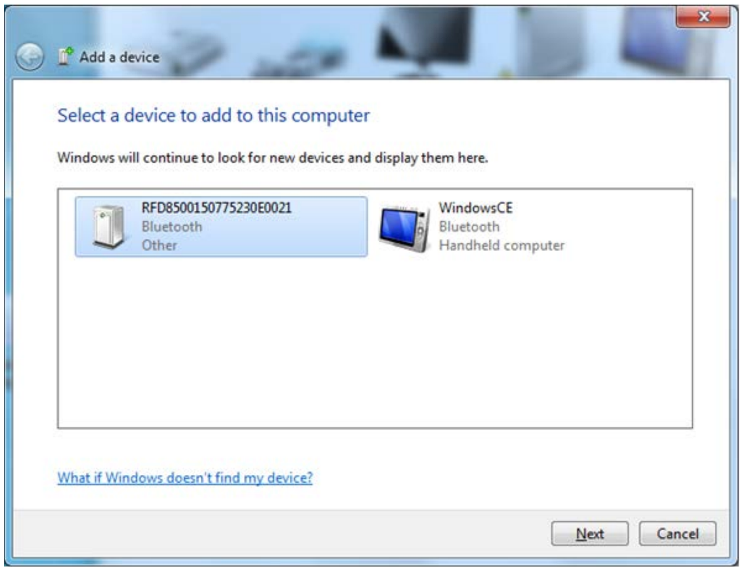
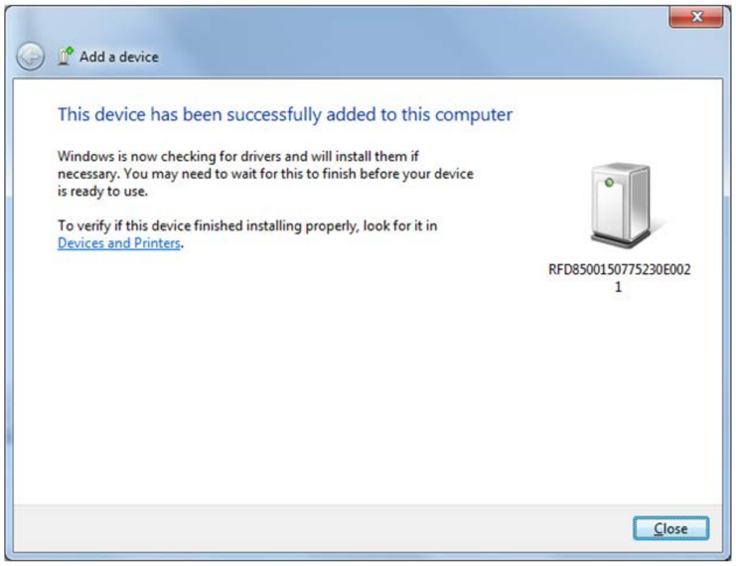

Add a device
Select a device to add to this computer
Windows will continue to look for new devices and display them here.
RFD8500150775230E0021
Bluetooth
Other
WindowsCE
Bluetooth
Handheld computer
What if Windows doesn't find my device?
Next
Cancel

Figure 1-2 *Adding a Device to Pair*

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



Add a device

This device has been successfully added to this computer

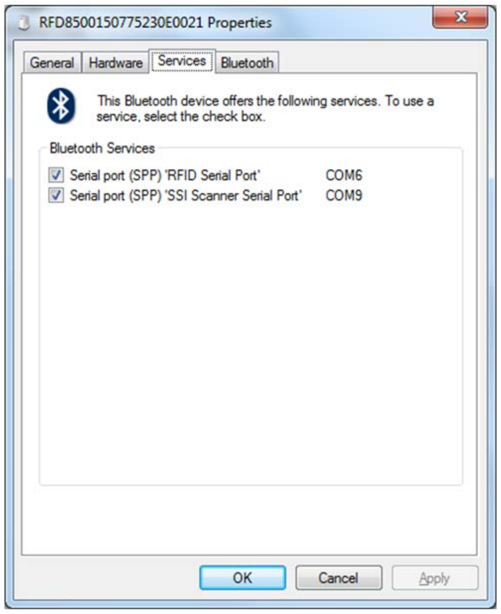
Windows is now checking for drivers and will install them if necessary. You may need to wait for this to finish before your device is ready to use.

To verify if this device finished installing properly, look for it in Devices and Printers.

RFD8500150775230E0021

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.



RFD8500150775230E0021 Properties

This Bluetooth device offers the following services. To use a service, select the check box.

Bluetooth Services

✓ Serial port (SPP) 'RFID Serial Port'	COM6
✓ Serial port (SPP) 'SSI Scanner Serial Port'	COM9

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.

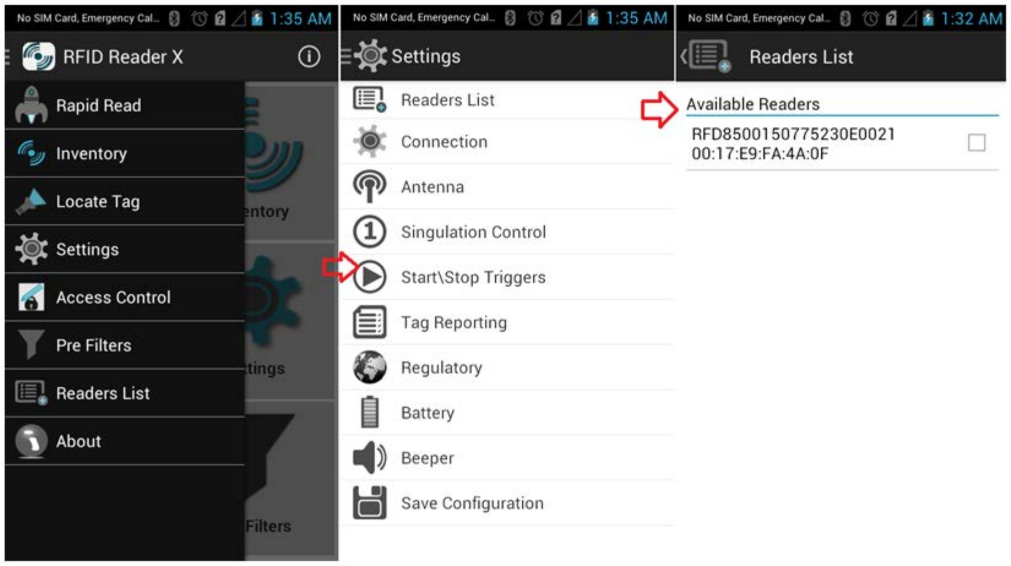


Figure 1-5 *Readers List*

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.

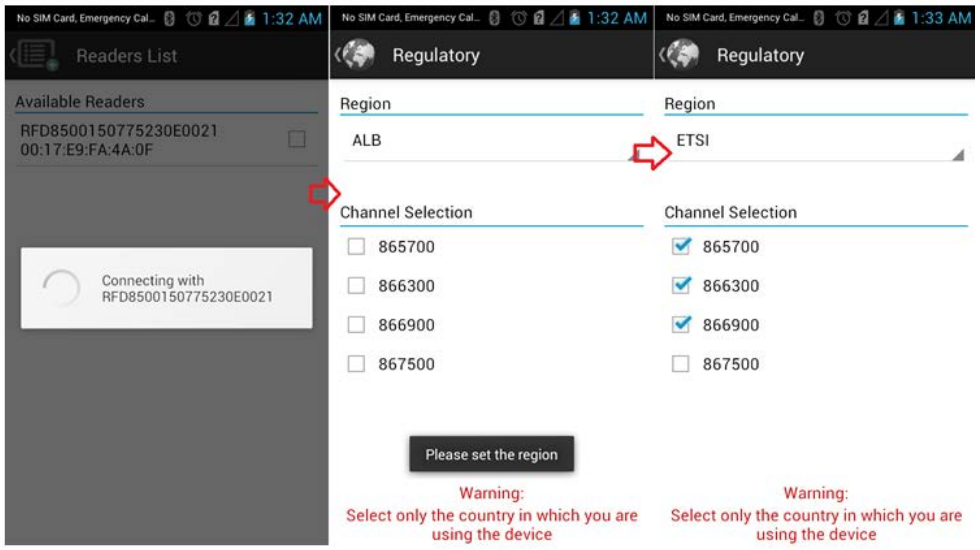


Figure 1-6 *Regulatory Information*

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.

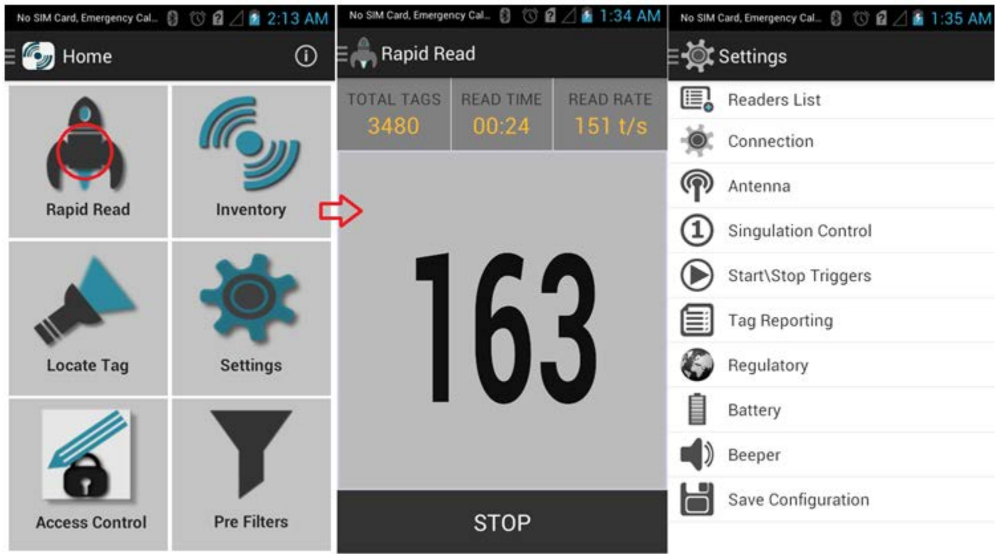


Figure 1-7 *Rapid Read Screen*

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

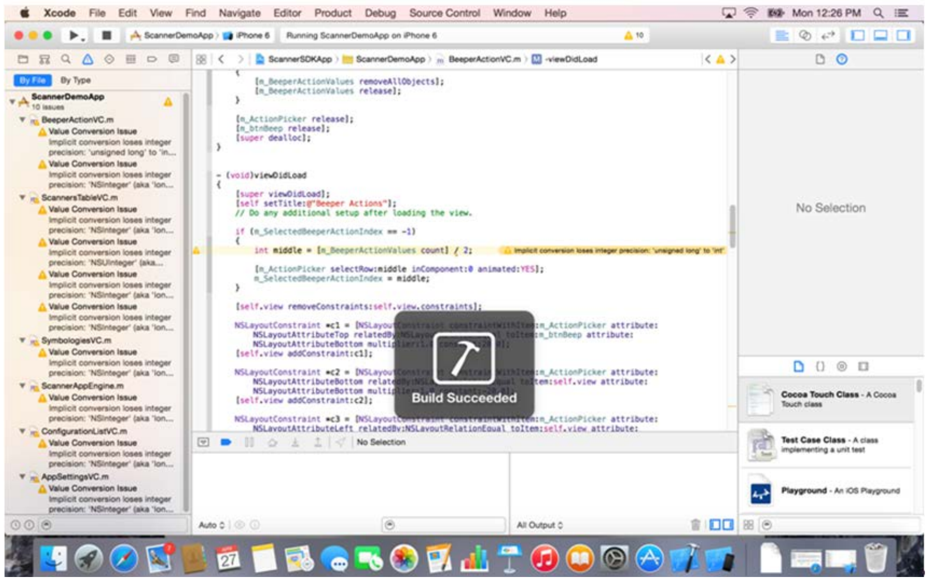
Using the Zebra RFID Mobile Application for iOS with iOS Devices

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 5 Touch; iPhone 6.

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


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
Mon 12:26 PM
ScannerDemoApp › iPhone 6
Running ScannerDemoApp on iPhone 6
ScannerSDKApp › ScannerDemoApp › BeeperActionVC.m › -viewDidLoad
No Selection
Build Succeeded
Cocoa Touch Class - A Cocoa Touch class
Test Case Class - A class implementing a unit test
Playground - An iOS Playground
Auto
All Output

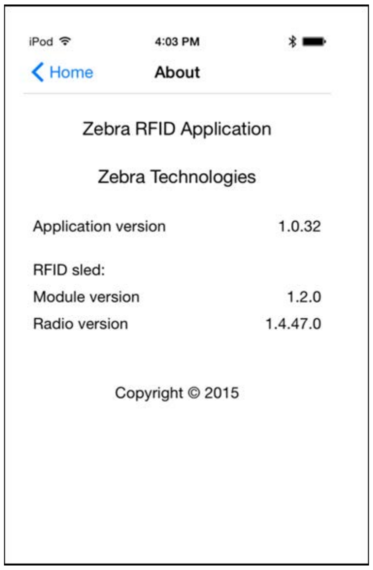
```
    {
        [m_BeeperActionValues removeAllObjects];
        [m_BeeperActionValues release];
    }

    [m_ActionPicker release];
    [m_btnBeep release];
    [super dealloc];
}

- (void)viewDidLoad
{
    [super viewDidLoad];
    [self setTitle:@"Beeper Actions"];
    // Do any additional setup after loading the view.

    if (m_SelectedBeeperActionIndex == -1)
    {
        int middle = [m_BeeperActionValues count] / 2;      Implicit conversion loses integer precision: 'unsigned long' to 'int'

        [m_ActionPicker selectRow:middle inComponent:0 animated:YES];
        m_SelectedBeeperActionIndex = middle;
    }

    [self.view removeConstraints:self.view.constraints];

    NSLayoutConstraint *c1 = [NSLayout[illegible]:m_ActionPicker attribute:
        NSLayoutAttributeTop relatedBy[illegible] toItem:m_btnBeep attribute:
        NSLayoutAttributeBottom multip[illegible]];
    [self.view addConstraint:c1];

    NSLayoutConstraint *c2 = [NSLayout[illegible]WithItem:m_ActionPicker attribute:
        NSLayoutAttributeBottom relat[illegible] toItem:self.view attribute:
        NSLayoutAttributeBottom multip[illegible];
    [self.view addConstraint:c2];

    NSLayoutConstraint *c3 = [NSLayout[illegible]:m_ActionPicker attribute:
        NSLayoutAttributeLeft relatedBy:NSLayoutRelationEqual toItem:self.view attribute:
```

Figure 1-8 *Successful Build Screen*

Pairing with an iOS Phone

See *Pairing with an Android Phone on page 1-3* 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.



Zebra RFID Application

Zebra Technologies

Application version	1.0.32
RFID sled:	
Module version	1.2.0
Radio version	1.4.47.0

Copyright © 2015

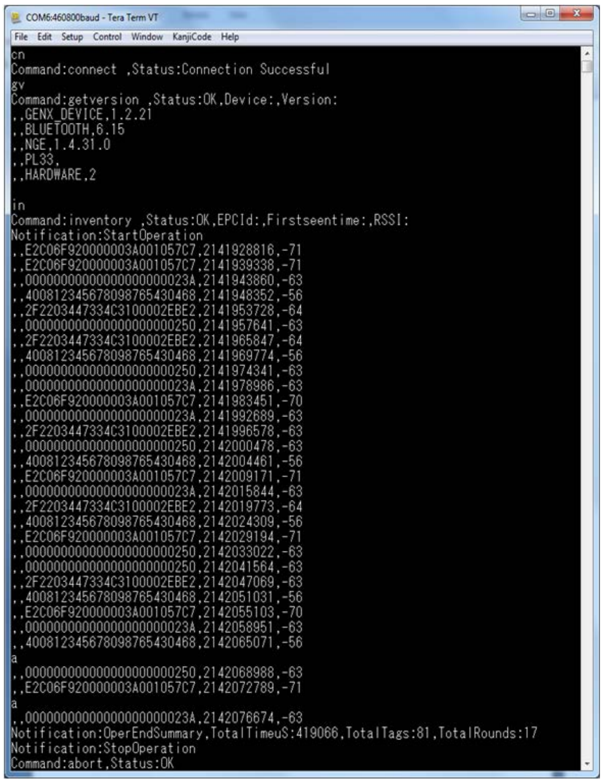
Figure 1-9 *Version Screen*

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 1-5*.
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.


COM6:460800baud - Tera Term VT
File Edit Setup Control Window KanjiCode Help

```
cn
Command:connect ,Status:Connection Successful
gv
Command:getversion ,Status:OK,Device:,Version:
,,GENX_DEVICE,1.2.21
,,BLUETOOTH,6.15
,,NGE,1.4.31.0
,,PL33,
,,HARDWARE,2

in
Command:inventory ,Status:OK,EPCId:,Firstseentime:,RSSI:
Notification:StartOperation
,,E2C06F920000003A001057C7,2141928816,-71
,,E2C06F920000003A001057C7,2141939338,-71
,,00000000000000000000023A,2141943860,-63
,,400812345678098765430468,2141948352,-56
,,2F2203447334C3100002EBE2,2141953728,-64
,,000000000000000000000250,2141957641,-63
,,2F2203447334C3100002EBE2,2141965847,-64
,,400812345678098765430468,2141969774,-56
,,000000000000000000000250,2141974341,-63
,,00000000000000000000023A,2141978986,-63
,,E2C06F920000003A001057C7,2141983451,-70
,,00000000000000000000023A,2141992689,-63
,,2F2203447334C3100002EBE2,2141996578,-63
,,000000000000000000000250,2142000478,-63
,,400812345678098765430468,2142004461,-56
,,E2C06F920000003A001057C7,2142009171,-71
,,00000000000000000000023A,2142015844,-63
,,2F2203447334C3100002EBE2,2142019773,-64
,,400812345678098765430468,2142024309,-56
,,E2C06F920000003A001057C7,2142029194,-71
,,000000000000000000000250,2142033022,-63
,,000000000000000000000250,2142041564,-63
,,2F2203447334C3100002EBE2,2142047069,-63
,,400812345678098765430468,2142051031,-56
,,E2C06F920000003A001057C7,2142055103,-70
,,00000000000000000000023A,2142058951,-63
,,400812345678098765430468,2142065071,-56
a
,,000000000000000000000250,2142068988,-63
,,E2C06F920000003A001057C7,2142072789,-71
a
,,00000000000000000000023A,2142076674,-63
Notification:OperEndSummary,TotalTimeuS:419066,TotalTags:81,TotalRounds:17
Notification:StopOperation
Command:abort,Status:OK
```

Figure 1-10 *Commands*

Chapter 2 GETTING STARTED with the ZEBRA RFID SDK for iOS

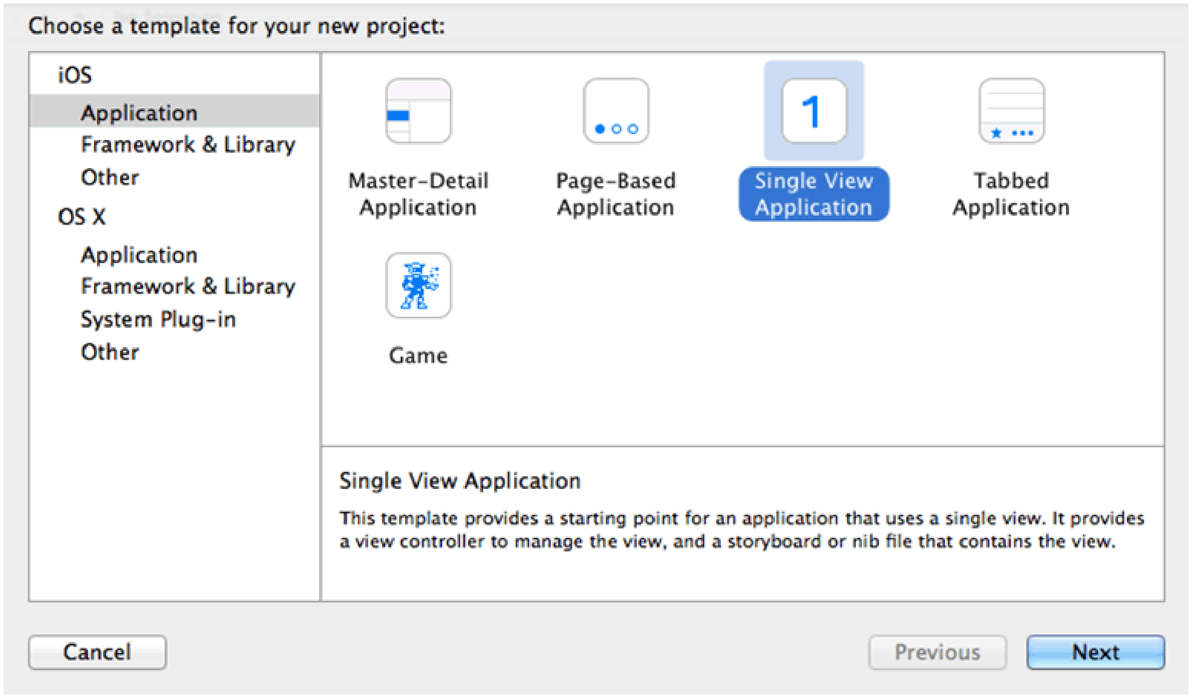
Introduction

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.

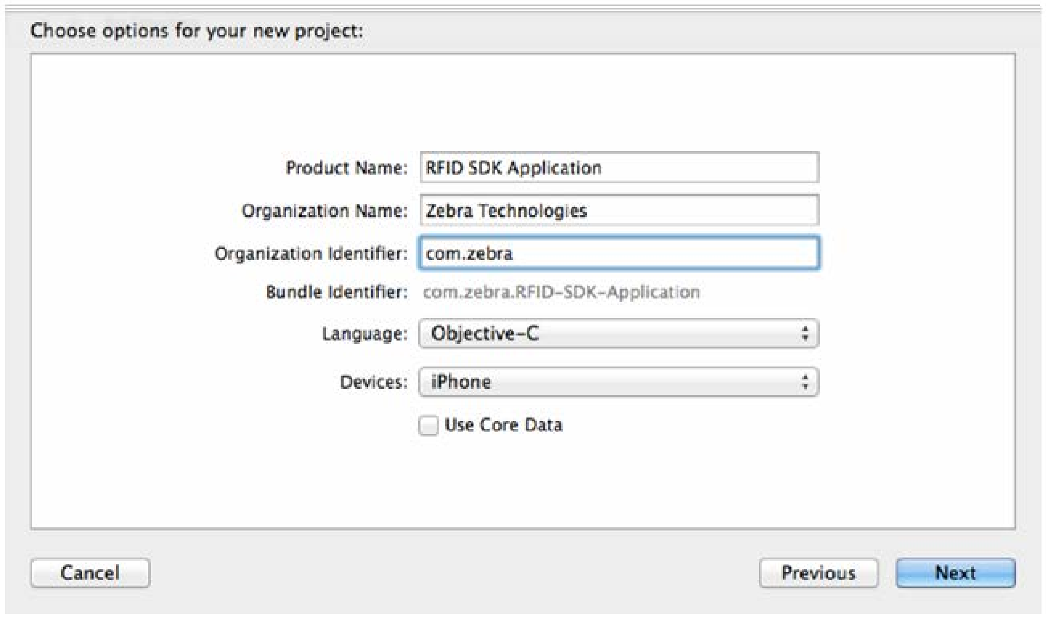


Single View Application

This template provides a starting point for an application that uses a single view. It provides a view controller to manage the view, and a storyboard or nib file that contains the view.

Figure 2-1 *Choosing Project Template*

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



Product Name: RFID SDK Application
Organization Name: Zebra Technologies
Organization Identifier: com.zebra
Bundle Identifier: com.zebra.RFID-SDK-Application
Language: Objective-C
Devices: iPhone
Use Core Data

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.

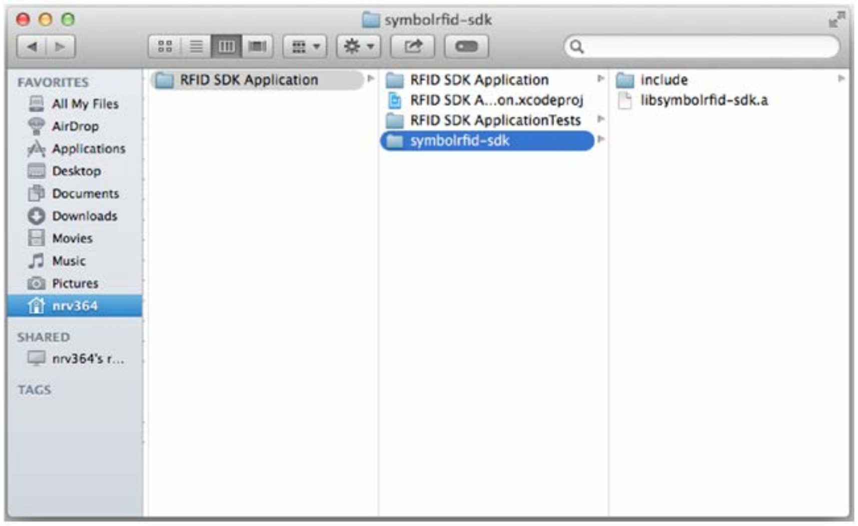
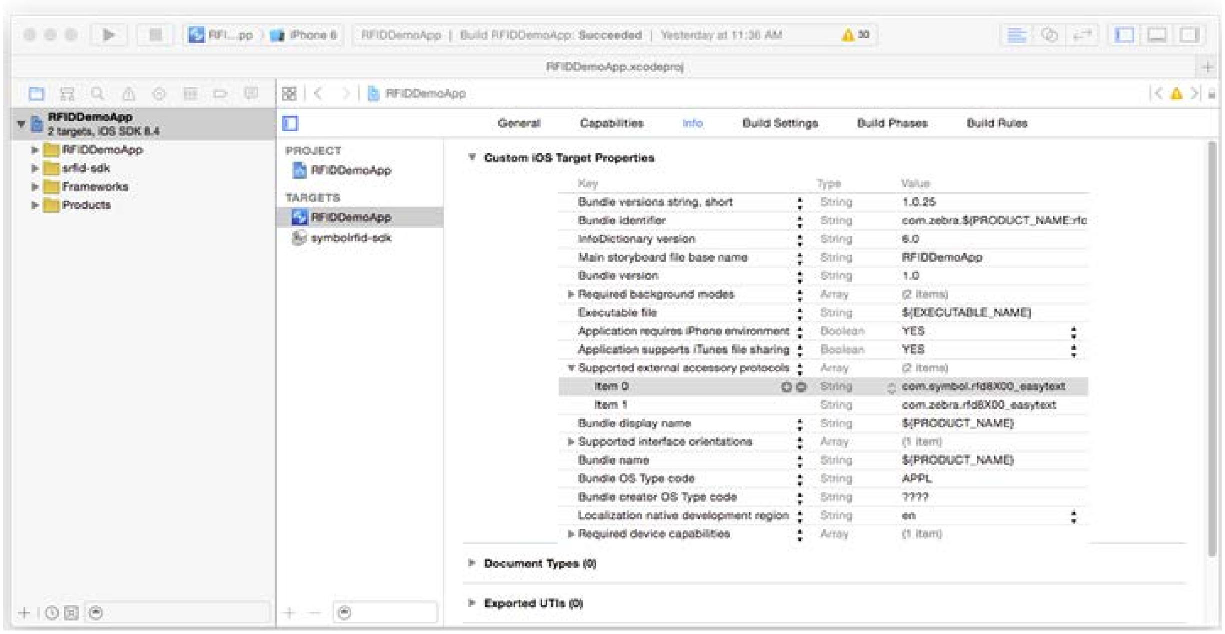


Figure 2-3 *Copying Folder to Project*

5. Configure your XCode project to support the com.zebra.rfd8x00_easytext and com.symbol.rfd8X00_easytext external accessory communication protocols by including the *UISupportedExternalAccessoryProtocols* key in your application's *Info.plist* file, or via the *[Info]* tab of your project settings.



Custom iOS Target Properties

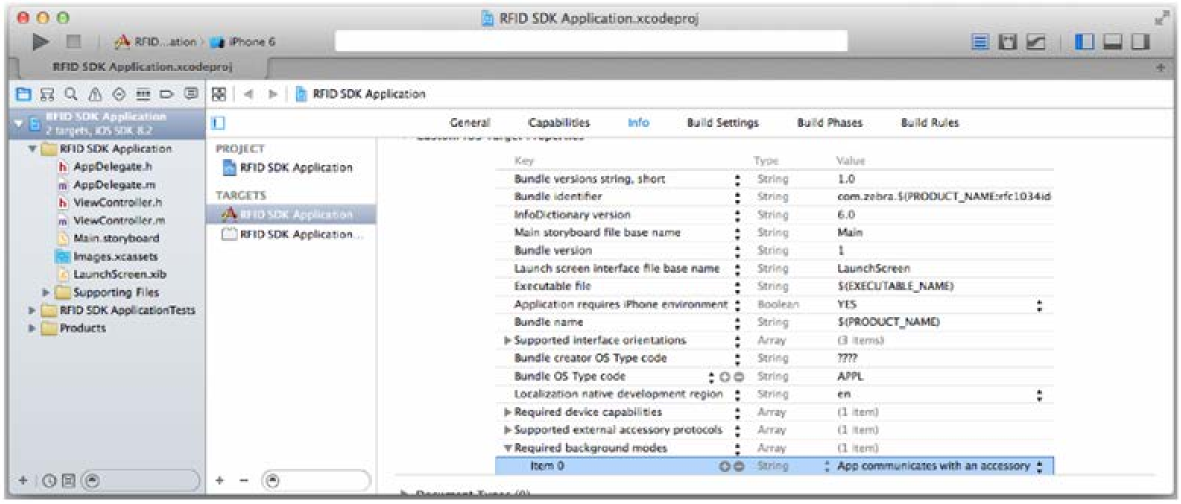
Key	Type	Value
Bundle versions string, short	String	1.0.25
Bundle identifier	String	com.zebra.$(PRODUCT_NAME:rfc
InfoDictionary version	String	6.0
Main storyboard file base name	String	RFIDDemoApp
Bundle version	String	1.0
Required background modes	Array	(2 items)
Executable file	String	$(EXECUTABLE_NAME)
Application requires iPhone environment	Boolean	YES
Application supports iTunes file sharing	Boolean	YES
Supported external accessory protocols	Array	(2 items)
Item 0	String	com.symbol.rfd8X00_easytext
Item 1	String	com.zebra.rfd8X00_easytext
Bundle display name	String	$(PRODUCT_NAME)
Supported interface orientations	Array	(1 item)
Bundle name	String	$(PRODUCT_NAME)
Bundle OS Type code	String	APPL
Bundle creator OS Type code	String	????
Localization native development region	String	en
Required device capabilities	Array	(1 item)

Document Types (0)

Exported UTIs (0)

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.



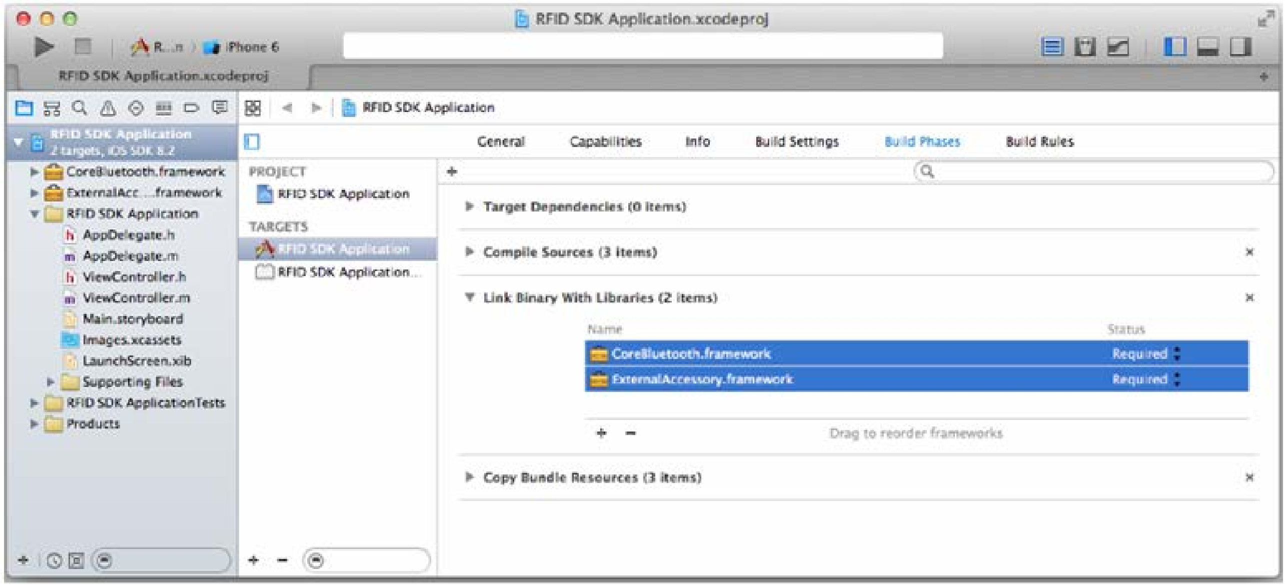
Key	Type	Value
Bundle versions string, short	String	1.0
Bundle identifier	String	com.zebra.$(PRODUCT_NAME:rfc1034id
InfoDictionary version	String	6.0
Main storyboard file base name	String	Main
Bundle version	String	1
Launch screen interface file base name	String	LaunchScreen
Executable file	String	$(EXECUTABLE_NAME)
Application requires iPhone environment	Boolean	YES
Bundle name	String	$(PRODUCT_NAME)
▶ Supported interface orientations	Array	(3 items)
Bundle creator OS Type code	String	????
Bundle OS Type code	String	APPL
Localization native development region	String	en
▶ Required device capabilities	Array	(1 item)
▶ Supported external accessory protocols	Array	(1 item)
▼ Required background modes	Array	(1 item)
Item 0	String	App communicates with an accessory

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

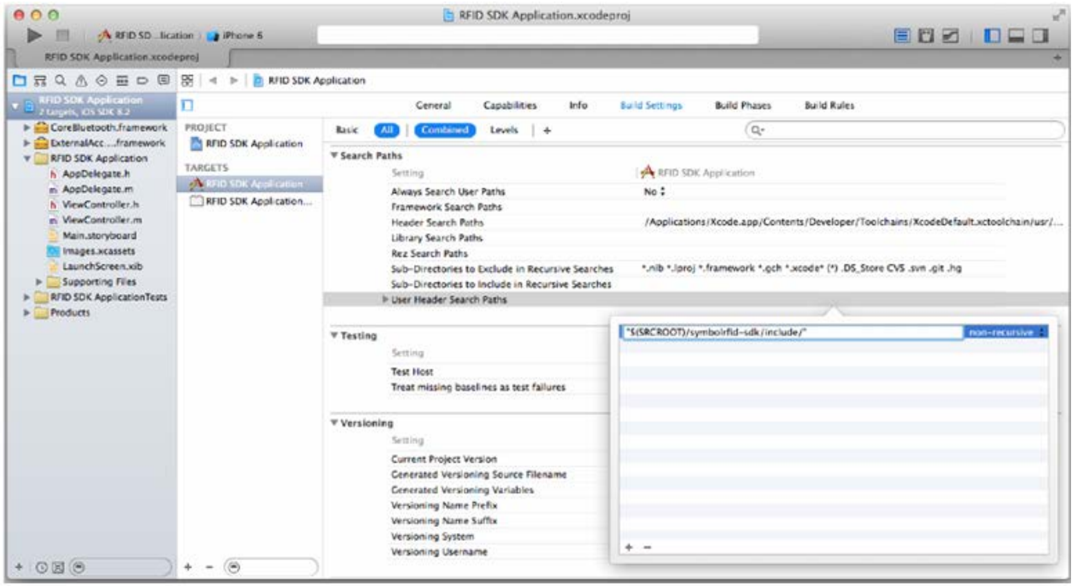
   - CoreBluetooth.framework



Name	Status
CoreBluetooth.framework	Required
ExternalAccessory.framework	Required

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.



▼ Search Paths

Setting	RFID SDK Application
Always Search User Paths	No
Framework Search Paths	
Header Search Paths	/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/...
Library Search Paths	
Rez Search Paths	
Sub-Directories to Exclude in Recursive Searches	*.nib *.lproj *.framework *.gch *.xcode* (*) .DS_Store CVS .svn .git .hg
Sub-Directories to Include in Recursive Searches	
User Header Search Paths	"$(SRCROOT)/symbolrfid-sdk/include/" non-recursive

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.

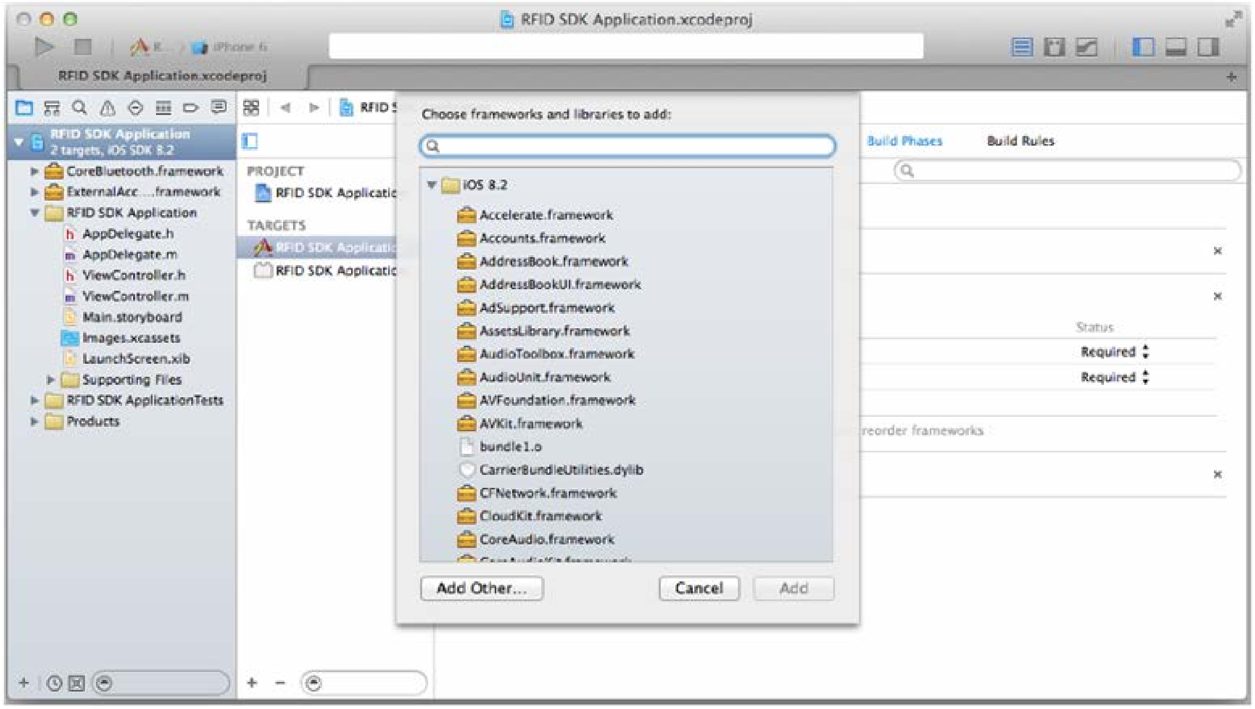


Figure 2-8 *Link Application to the Zebra RFID SDK for iOS Static Library*

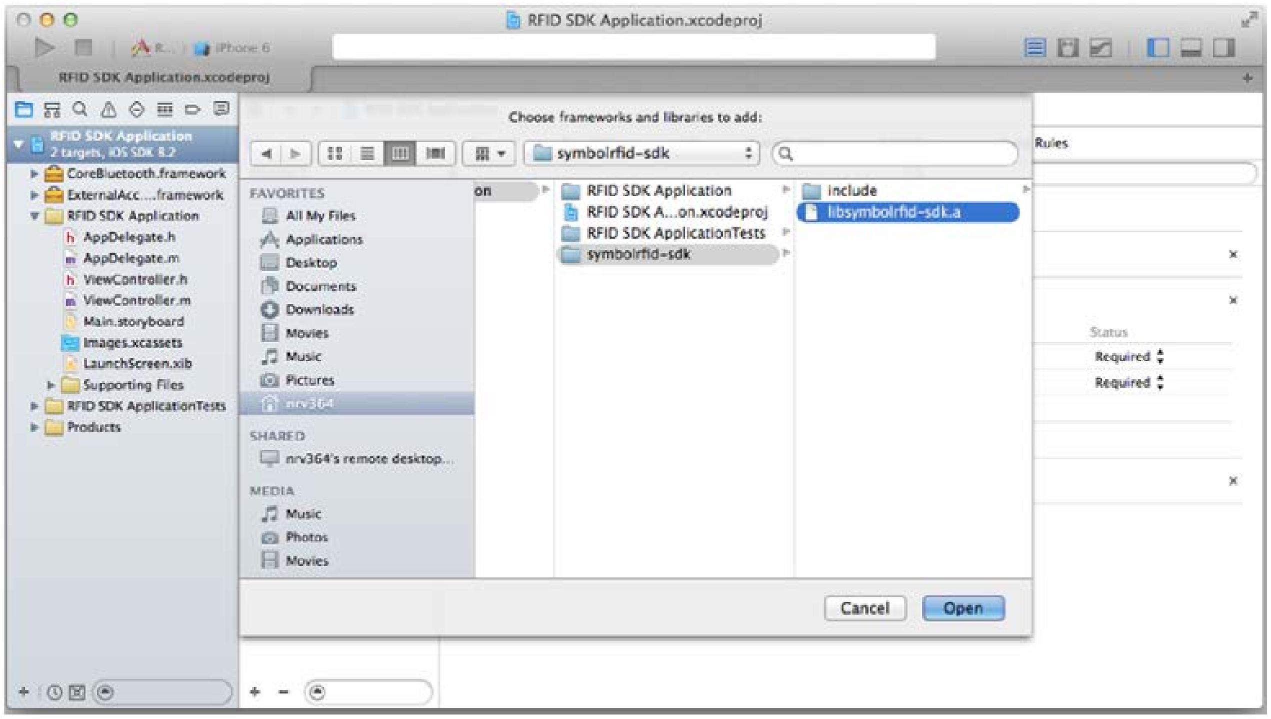


Figure 2-9 *Select the Static Library - libsymbolrfid-sdk.a*

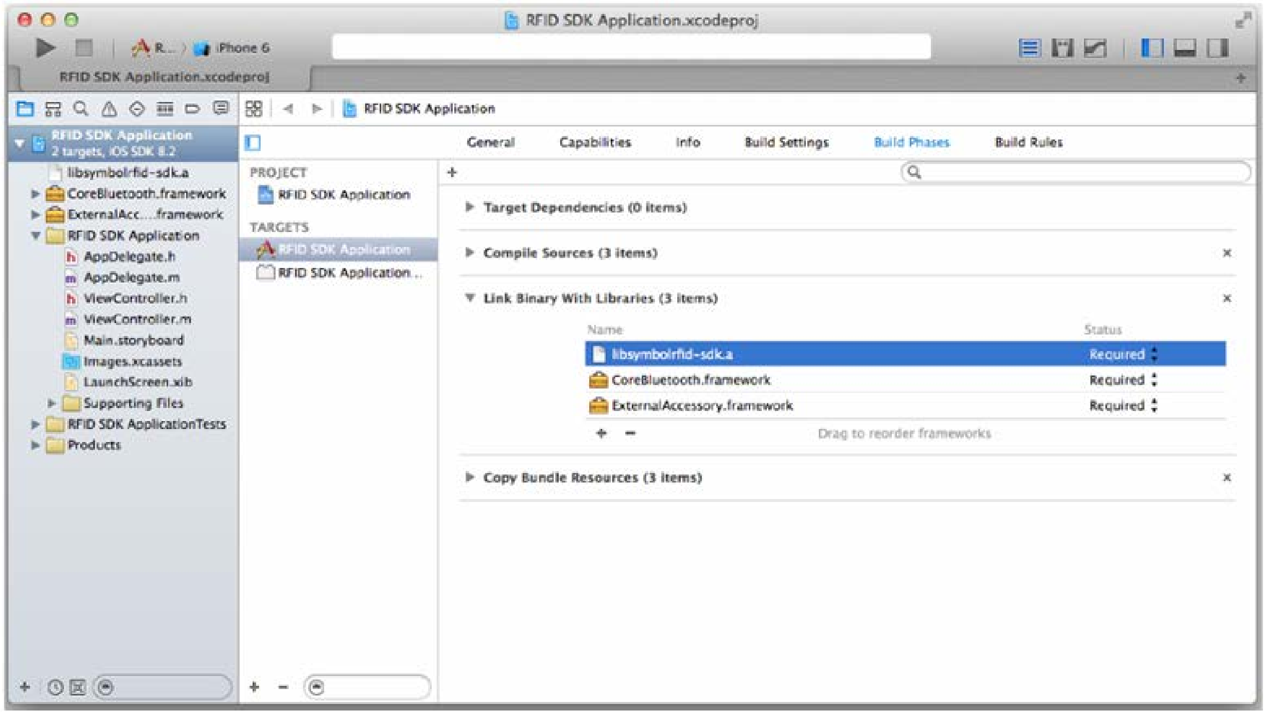

RFID SDK Application.xcodeproj
General
Capabilities
Info
Build Settings
Build Phases
Build Rules
PROJECT
TARGETS
Target Dependencies (0 items)
Compile Sources (3 items)
Link Binary With Libraries (3 items)
Name
Status
libsymbolrfid-sdk.a Required
CoreBluetooth.framework Required
ExternalAccessory.framework Required
Drag to reorder frameworks
Copy Bundle Resources (3 items)

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

Chapter 3 ZEBRA RFID SDK for iOS API

Introduction

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

Application Programming Interface Definition

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

Constants

Result Codes

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

Table 3-1 *Constants - Result Codes*

Result Code	Description	Value
SRFID_RESULT_SUCCESS	A specific API function has completed successfully	0
SRFID_RESULT_FAILURE	A specific API function has completed unsuccessfully.	1
SRFID_RESULT_READER_NOT_AVAILABLE	A specific API function has completed unsuccessfully because a specified RFID reader was not available/active.	2
SRFID_RESULT_INVALID_PARAMS	A specific API function has completed unsuccessfully due to invalid input and/or output parameters.	4
SRFID_RESULT_RESPONSE_TIMEOUT	A specific API function has completed unsuccessfully due to expiration of a response timeout during communication with a specific RFID reader.	5
SRFID_RESULT_NOT_SUPPORTED	A specific API function is not supported by the SDK.	6
SRFID_RESULT_ RESPONSE_ERROR	An error response has been received from RFID reader.	7
SRFID_RESULT_WRONG_ASCII_PASSWORD	Incorrect password has been provided for ASCII protocol level connection establishment.	8
SRFID_RESULT_ASCII_CONNECTION_REQUIRED	An ASCII protocol level connection required for communication with RFID reader was not established.	9

Operating Modes

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

Table 3-2 *Constants - Operating Modes*

Operating Mode	Description	Value
SRFID_OPMODE_MFI	The SDK is able to communicate with RFID readers in "iOS BT MFi" mode only.	1
SRFID_OPMODE_BTLE	The SDK is able to communicate with RFID readers in "iOS BT LE" mode only.	2
SRFID_OPMODE_ALL	The SDK is able to communicate with RFID readers in "iOS BT MFi" mode and with RFID readers in "iOS BT LE" mode.	3

RFID Reader Modes

These constants are defined to represent communication modes of RFID readers.

Table 3-3 *Constants - RFID Reader Modes*

Scanner Mode	Description	Value
SRFID_CONNTYPE_INVALID	The SDK is unable to determine communication mode of a specific RFID reader.	0
SRFID_CONNTYPE_MFI	A specific RFID reader is in "iOS BT MFi" mode.	1
SRFID_CONNTYPE_BTLE	A specific RFID reader is in "iOS BT LE" mode.	2

Notifications

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

Table 3-4 *Constants - Notifications*

Notification	Description	Value
SRFID_EVENT_READER_APPEARANCE	"Device Arrival" notification (appearance of an available RFID reader).	2
SRFID_EVENT_READER_DISAPPEARANCE	"Device Disappeared" notification (disappearance of an available RFID reader).	4
SRFID_EVENT_SESSION_ESTABLISHMENT	"Session Established" notification (appearance of a specific active RFID reader).	8
SRFID_EVENT_SESSION_TERMINATION	"Session Terminated" notification (disappearance of an active RFID reader).	16
SRFID_EVENT_MASK_READ	"Read Event" notification (reception of RFID tag related data from a specific active RFID reader).	32

Table 3-4 *Constants - Notifications (Continued)*

Notification	Description	Value
SRFID_EVENT_MASK_STATUS	"Status Event" notification (reception of a specific notification related to starting/stopping of a specific operation from a specific active RFID reader).	64
SRFID_EVENT_MASK_PROXIMITY	"Proximity Event" notification (reception of a specific proximity notification during ongoing tag locationing operation from a specific active RFID reader).	128
SRFID_EVENT_MASK_TRIGGER	"Trigger Event" notification (reception of a specific notification related to pressing/releasing a hand-held trigger from a specific active RFID reader).	256
SRFID_EVENT_MASK_BATTERY	"Battery Event" notification (reception of a specific battery information related notification from a specific active RFID reader).	512
SRFID_EVENT_MASK_STATUS_OPERENDSUMMARY	"Status Event" notification (reception of a Operation end summary notification from a specific active RFID reader).	1024

RFID Reader Models

These constants are defined to represent models of RFID readers supported by the Zebra RFID SDK for iOS.

Table 3-5 *Constants - RFID Reader Models*

Reader Models	Description	Value
SRFID_DEVMODEL_INVALID	The model either unknown, not recognized or not supported.	0
SRFID_DEVMODEL_RFID_RFD8500	RFD8500 in RFID reader mode.	1

Status Notifications

These constants are defined to represent notifications related to operation status supported by the Zebra RFID SDK for iOS.

Table 3-6 *Constants - Status Notifications*

Status Notification	Description	Value
SRFID_EVENT_STATUS_OPERATION_START	Operation was started.	0
SRFID_EVENT_STATUS_OPERATION_STOP	Operation was terminated.	1
SRFID_EVENT_STATUS_OPERATION_BATCHMODE	Operation was started in batch mode.	2
SRFID_EVENT_STATUS_OPERATION_END_SUMMARY	Operation End Summary event.	3

Memory Bank Identifiers

These constants are defined to represent identifiers of memory banks supported by the Zebra RFID SDK for iOS.

Table 3-7 *Constants - Memory Bank Identifiers*

Memory Bank Identifier	Description	Value
SRFID_MEMORYBANK_EPC	EPC memory bank.	1
SRFID_MEMORYBANK_TID	TID memory bank.	2
SRFID_MEMORYBANK_USER	User memory bank.	3
SRFID_MEMORYBANK_RESV	Reserved memory bank.	4
SRFID_MEMORYBANK_NONE	Memory bank is not specified.	5
SRFID_MEMORYBANK_ACCESS	Access memory bank.	6
SRFID_MEMORYBANK_KILL	Kill memory bank.	7

Access Operation Codes

These constants are defined to represent access operations supported by the Zebra RFID SDK for iOS.

Table 3-8 *Constants - Access Operation Codes*

Access Operation Code	Description	Value
SRFID_ACCESSOPERATIONCODE_READ	Read access operation.	0
SRFID_ ACCESSOPERATIONCODE_WRITE	Write access operation.	1
SRFID_ ACCESSOPERATIONCODE_LOCK	Lock access operation.	2
SRFID_ ACCESSOPERATIONCODE_KILL	Kill access operation.	3
SRFID_ ACCESSOPERATIONCODE_BLOCK_WRITE	Block write access operation.	4
SRFID_ ACCESSOPERATIONCODE_BLOCK_ERASE	Block erase access operation.	5
SRFID_ ACCESSOPERATIONCODE_RECOMMISSION	Not supported.	6
SRFID_ ACCESSOPERATIONCODE_BLOCK_PERMALOCK	Block permanent lock access operation.	7
SRFID_ ACCESSOPERATIONCODE_NXP_SET_EAS	Not supported.	8
SRFID_ ACCESSOPERATIONCODE_NXP_READ_PROTECT	Not supported.	9
SRFID_ ACCESSOPERATIONCODE_NXP_RESET_READ_PROTECT	Not supported.	10
SRFID_ ACCESSOPERATIONCODE_NXP_CHANGE_CONFIG	Not supported.	22
SRFID_ ACCESSOPERATIONCODE_IMPINJ_QT_READ	Not supported.	21
SRFID_ ACCESSOPERATIONCODE_IMPINJ_QT_WRITE	Not supported.	20
SRFID_ ACCESSOPERATIONCODE_NONE	Access operation is not specified.	0xFF

Divide Ratio Codes

These constants are defined to represent divide ratio parameters of RF modes supported by the Zebra RFID SDK for iOS.

Table 3-9 *Constants - Divide Ratio Codes*

Divide Ratio Code	Description	Value
SRFID_DIVIDERATIO_DR_8		0
SRFID_DIVIDERATIO_DR_64_3		1

Modulation Codes

These constants are defined to represent modulation parameters of RF modes supported by the Zebra RFID SDK for iOS.

Table 3-10 *Constants - Modulation Codes*

Modulation Code	Description	Value
SRFID_MODULATION_MV_FM0		0
SRFID_MODULATION_MV_2		1
SRFID_MODULATION_MV_4		2
SRFID_MODULATION_MV_8		3

Forward Link Modulation Codes

These constants are defined to represent forward link modulation parameters of RF modes supported by the Zebra RFID SDK for iOS.

Table 3-11 *Constants - Forward Link Modulation Codes*

Forward Link Modulation Code	Description	Value
SRFID_FORWARDLINKMODULATION_PR_ASK		0
SRFID_FORWARDLINKMODULATION_SSB_ASK		1
SRFID_FORWARDLINKMODULATION_DSB_ASK		2

Spectral Mask Indicator Codes

These constants are defined to represent spectral mask indicator parameters of RF modes supported by the Zebra RFID SDK for iOS.

Table 3-12 *Constants - Spectral Mask Indicator Codes*

Spectral Mask Indicator Code	Description	Value
SRFID_SPECTRALMASKINDICATOR_SI		1
SRFID_SPECTRALMASKINDICATOR_MI		2
SRFID_SPECTRALMASKINDICATOR_DI		3

Singulation Selection Codes

These constants are defined to represent selection (SL flag) parameters of singulation configuration supported by the Zebra RFID SDK for iOS.

Table 3-13 *Constants - Singulation Selection Codes*

Singulation Selection Code	Description	Value
SRFID_SLFLAG_ASSERTED	Select asserted.	0
SRFID_SLFLAG_DEASSERTED	Select de-asserted.	1
SRFID_SLFLAG_ALL	Select all.	2

Singulation Session Codes

These constants are defined to represent session parameters of singulation configuration supported by Zebra RFID SDK for iOS.

Table 3-14 *Constants - Singulation Session Codes*

Singulation Session Code	Description	Value
SRFID_SESSION_S0	Session S0.	0
SRFID_SESSION_S1	Session S1.	1
SRFID_SESSION_S2	Session S2.	2
SRFID_SESSION_S3	Session S3.	3

Singulation Target Codes

These constants are defined to represent target (inventory state) parameters of singulation configuration supported by the Zebra RFID SDK for iOS.

Table 3-15 *Constants - Singulation Target Codes*

Singulation Target Code	Description	Value
SRFID_INVENTORYSTATE_A	Inventory state A.	0
SRFID_INVENTORYSTATE_B	Inventory state B.	1
SRFID_INVENTORYSTATE_AB_FLIP	Automatically repeat inventory with another query after flipping Target flag.	2

Trigger Type Codes

These constants are defined to represent types of hand-held triggers supported by the Zebra RFID SDK for iOS.

Table 3-16 *Constants - Trigger Type Codes*

Trigger Code	Description	Value
SRFID_TRIGGERTYPE_PRESS	Trigger press.	0
SRFID_TRIGGERTYPE_RELEASE	Trigger release.	1

Prefilter Target Codes

These constants are defined to represent target parameters of prefilter (select record) configuration supported by the Zebra RFID SDK for iOS.

Table 3-17 *Constants - Prefilter Target Codes*

Prefilter Target Code	Description	Value
SRFID_SELECTTARGET_S0	Session S0.	0
SRFID_SELECTTARGET_S1	Session S1.	1
SRFID_SELECTTARGET_S2	Session S2.	2
SRFID_SELECTTARGET_S3	Session S3.	3
SRFID_SELECTTARGET_SL	Select flag.	4

Prefilter Action Codes

These constants are defined to represent action parameters of prefilter (select record) configuration supported by the Zebra RFID SDK for iOS.

Table 3-18 *Constants - Prefilter Action Codes*

Prefilter Action Code	Description	Value
SRFID_SELECTACTION_INV_A_NOT_INV_B__OR__ASRT_SL_NOT_DSRT_SL		1
SRFID_SELECTACTION_INV_A__OR__ASRT_SL		2
SRFID_SELECTACTION_NOT_INV_B__OR__NOT_DSRT_SL		3
SRFID_SELECTACTION_INV_A2BB2A_NOT_INV_A__OR__NEG_SL_NOT_ASRT_SL		4
SRFID_SELECTACTION_INV_B_NOT_INV_A__OR__DSRT_SL_NOT_ASRT_SL		5
SRFID_SELECTACTION_INV_B__OR__DSRT_SL		6
SRFID_SELECTACTION_NOT_INV_A__OR__NOT_ASRT_SL		7
SRFID_SELECTACTION_NOT_INV_A2BB2A__OR__NOT_NEG_SL		8

Access Permission Codes

These constants are defined to represent permission (lock action) parameters of lock operation supported by the Zebra RFID SDK for iOS.

Table 3-19 *Constants - Access Permission Codes*

Access Permission Code	Description	Value
SRFID_ACCESSPERMISSION_ACCESSIBLE	Accessible, only from open or secure states.	0
SRFID_ACCESSPERMISSION_ACCESSIBLE_PERM	Permanently accessible, only from open or secure states.	1
SRFID_ACCESSPERMISSION_ACCESSIBLE_SECURED	Accessible, only from secure states.	2
SRFID_ACCESSPERMISSION_ALWAYS_NOT_ACCESSIBLE	Never accessible in any states.	3

Beeper Configuration Codes

These constants are defined to represent beeper related parameters supported by the Zebra RFID SDK for iOS.

Table 3-20 *Constants - Beeper Configuration Codes*

Access Permission Code	Description	Value
SRFID_BEEPERCONFIG_HIGH	Beeper enabled, high volume.	0
SRFID_BEEPERCONFIG_MEDIUM	Beeper enabled, medium volume.	1
SRFID_BEEPERCONFIG_LOW	Beeper enabled, low volume.	2
SRFID_BEEPERCONFIG_QUIET	Beeper disabled.	3

Trigger Event Codes

These constants are defined to represent events related to a hand-held trigger supported by Zebra RFID SDK for iOS.

Table 3-21 *Constants - Trigger Event Codes*

Trigger Code	Description	Value
SRFID_TRIGGEREVENT_PRESSED	A hand-held trigger was pressed.	0
SRFID_TRIGGEREVENT_RELEASED	A hand-held trigger was released.	1

Hopping Configuration Codes

These constants are defined to represent available options for hopping parameters in the regulatory configuration supported by the Zebra RFID SDK for iOS.

Table 3-22 *Constants - Hopping Configuration Codes*

Trigger Code	Description	Value
SRFID_HOPPINGCONFIG_DEFAULT	Hopping is not changed during application of regulatory configuration.	0
SRFID_HOPPINGCONFIG_ENABLED	Hopping is enabled during application of regulatory configuration.	1
SRFID_HOPPINGCONFIG_DISABLED	Hopping is disabled during application of regulatory configuration.	2

BatchMode Configuration Codes

These constants are defined to represent batch mode related parameters supported by the Zebra RFID SDK for iOS.

Table 3-23 *Constants - BatchMode Configuration Codes*

Trigger Code	Description	Value
SRFID_BATCHMODECONFIG_DISABLE	BatchMode is disabled.	0
SRFID_BATCHMODECONFIG_AUTO	BatchMode is set as auto.	1
SRFID_BATCHMODECONFIG_ENABLE	BatchMode is enabled.	2

Types

SRFID_RESULT

Description

Type of return value of the Zebra RFID SDK for iOS API functions.

```
typedef enum {
    SRFID_RESULT_SUCCESS                   = 0x00,
    SRFID_RESULT_FAILURE                   = 0x01,
    SRFID_RESULT_READER_NOT_AVAILABLE      = 0x02,
    SRFID_RESULT_INVALID_PARAMS            = 0x04,
    SRFID_RESULT_RESPONSE_TIMEOUT          = 0x05,
    SRFID_RESULT_NOT_SUPPORTED             = 0x06,
    SRFID_RESULT_RESPONSE_ERROR            = 0x07,
    SRFID_RESULT_WRONG_ASCII_PASSWORD      = 0x08,
    SRFID_RESULT_ASCII_CONNECTION_REQUIRED = 0x09
} SRFID_RESULT;
```

SRFID_EVENT_STATUS

Description

Notification codes related to operation status supported by the Zebra RFID SDK for iOS.

```
typedef enum {
    SRFID_EVENT_STATUS_OPERATION_START       = 0x00,
    SRFID_EVENT_STATUS_OPERATION_STOP        = 0x01,
    SRFID_EVENT_STATUS_OPERATION_BATCHMODE   = 0x02,
    SRFID_EVENT_STATUS_OPERATION_END_SUMMARY = 0x03
} SRFID_EVENT_STATUS;
```

SRFID_MEMORYBANK

Description

Used to specify one of memory banks supported by the Zebra RFID SDK for iOS.

```
typedef enum {
    SRFID_MEMORYBANK_EPC                    = 0x01,
    SRFID_MEMORYBANK_TID                    = 0x02,
    SRFID_MEMORYBANK_USER                   = 0x03,
    SRFID_MEMORYBANK_RESV                   = 0x04,
    SRFID_MEMORYBANK_NONE                   = 0x05,
    SRFID_MEMORYBANK_ACCESS                 = 0x06,
    SRFID_MEMORYBANK_KILL                   = 0x07
} SRFID_MEMORYBANK;
```

SRFID_ACCESSOPERATIONCODE

Description

Used to specify one of access operations supported by the Zebra RFID SDK for iOS.

```
typedef enum
{
    SRFID_ACCESSOPERATIONCODE_READ                              = 0,
    SRFID_ACCESSOPERATIONCODE_WRITE                             = 1,
    SRFID_ACCESSOPERATIONCODE_LOCK                              = 2,
    SRFID_ACCESSOPERATIONCODE_KILL                              = 3,
    SRFID_ACCESSOPERATIONCODE_BLOCK_WRITE                       = 4,
    SRFID_ACCESSOPERATIONCODE_BLOCK_ERASE                       = 5,
    SRFID_ACCESSOPERATIONCODE_RECOMMISSION                      = 6,
    SRFID_ACCESSOPERATIONCODE_BLOCK_PERMALOCK                   = 7,
    SRFID_ACCESSOPERATIONCODE_NXP_SET_EAS                       = 8,
    SRFID_ACCESSOPERATIONCODE_NXP_READ_PROTECT                  = 9,
    SRFID_ACCESSOPERATIONCODE_NXP_RESET_READ_PROTECT            = 10,
    SRFID_ACCESSOPERATIONCODE_NXP_CHANGE_CONFIG                 = 22,
    SRFID_ACCESSOPERATIONCODE_IMPINJ_QT_READ                    = 21,
    SRFID_ACCESSOPERATIONCODE_IMPINJ_QT_WRITE                   = 20,
    SRFID_ACCESSOPERATIONCODE_NONE                              = 0xFF,
} SRFID_ACCESSOPERATIONCODE;
```

SRFID_DIVIDERATIO

Description

Used to represent a value of divide ratio parameter of a specific RF mode.

```
typedef enum
{
    SRFID_DIVIDERATIO_DR_8                  = 0,
    SRFID_DIVIDERATIO_DR_64_3               = 1,
} SRFID_DIVIDERATIO;
```

SRFID_MODULATION

Description

Used to represent a value of modulation parameter of a specific RF mode.

```
typedef enum
{
    SRFID_MODULATION_MV_FM0                 = 0,
    SRFID_MODULATION_MV_2                   = 1,
    SRFID_MODULATION_MV_4                   = 2,
    SRFID_MODULATION_MV_8                   = 3,
} SRFID_MODULATION;
```

SRFID_FORWARDLINKMODULATION

Description

Used to represent a value of forward link modulation option of a specific RF mode.

```
typedef enum
{
    SRFID_FORWARDLINKMODULATION_PR_ASK      = 0,
    SRFID_FORWARDLINKMODULATION_SSB_ASK     = 1,
    SRFID_FORWARDLINKMODULATION_DSB_ASK     = 2,
} SRFID_FORWARDLINKMODULATION;
```

SRFID_SPECTRALMASKINDICATOR

Description

Used to represent a value of spectral mask indicator parameter of a specific RF mode.

```
typedef enum
{
    SRFID_SPECTRALMASKINDICATOR_SI          = 1,
    SRFID_SPECTRALMASKINDICATOR_MI          = 2,
    SRFID_SPECTRALMASKINDICATOR_DI          = 3,
} SRFID_SPECTRALMASKINDICATOR;
```

SRFID_SLFLAG

Description

Used to represent a value of selection (SL flag) parameter of singulation configuration.

```
typedef enum
{
    SRFID_SLFLAG_ASSERTED                   = 0,
    SRFID_SLFLAG_DEASSERTED                 = 1,
    SRFID_SLFLAG_ALL                        = 2,
} SRFID_SLFLAG;
```

SRFID_SESSION

Description

Used to represent a value of session parameter of singulation configuration.

```
typedef enum
{
    SRFID_SESSION_S0                        = 0,
    SRFID_SESSION_S1                        = 1,
    SRFID_SESSION_S2                        = 2,
    SRFID_SESSION_S3                        = 3,
} SRFID_SESSION;;
```

SRFID_INVENTORYSTATE

Description

Used to represent a value of inventory state (target) parameter of singulation configuration.

```
typedef enum
{
    SRFID_INVENTORYSTATE_A                  = 0,
    SRFID_INVENTORYSTATE_B                  = 1,
    SRFID_INVENTORYSTATE_AB_FLIP            = 2,
} SRFID_INVENTORYSTATE;
```

SRFID_TRIGGERTYPE

Description

Used to represent a types of hand-held triggers.

```
typedef enum {
    SRFID_TRIGGERTYPE_PRESS                 = 0x00,
    SRFID_TRIGGERTYPE_RELEASE               = 0x01,
} SRFID_TRIGGERTYPE;
```

SRFID_SELECTTARGET

Description

Used to represent a value of target parameter of prefilter (select record) configuration.

```
typedef enum {
    SRFID_SELECTTARGET_S0                      = 0x00,
    SRFID_SELECTTARGET_S1                      = 0x01,
    SRFID_SELECTTARGET_S2                      = 0x02,
    SRFID_SELECTTARGET_S3                      = 0x03,
    SRFID_SELECTTARGET_SL                      = 0x04,
} SRFID_SELECTTARGET;
```

SRFID_SELECTACTION

Description

Used to represent a value of action parameter of prefilter (select record) configuration.

```
typedef enum {
    SRFID_SELECTACTION_INV_A_NOT_INV_B__OR__ASRT_SL_NOT_DSRT_SL       = 0x00,
    SRFID_SELECTACTION_INV_A__OR__ASRT_SL                             = 0x01,
    SRFID_SELECTACTION_NOT_INV_B__OR__NOT_DSRT_SL                     = 0x02,
    SRFID_SELECTACTION_INV_A2BB2A_NOT_INV_A__OR__NEG_SL_NOT_ASRT_SL   = 0x03,
    SRFID_SELECTACTION_INV_B_NOT_INV_A__OR__DSRT_SL_NOT_ASRT_SL       = 0x04,
    SRFID_SELECTACTION_INV_B__OR__DSRT_SL                             = 0x05,
    SRFID_SELECTACTION_NOT_INV_A__OR__NOT_ASRT_SL                     = 0x06,
    SRFID_SELECTACTION_NOT_INV_A2BB2A__OR__NOT_NEG_SL                 = 0x07,
} SRFID_SELECTACTION;
```

SRFID_ACCESSPERMISSION

Description

Used to represent a value of permission (lock action) parameter of lock operation.

```
typedef enum {
    SRFID_ACCESSPERMISSION_ACCESSIBLE                 = 0x00,
    SRFID_ACCESSPERMISSION_ACCESSIBLE_PERM            = 0x01,
    SRFID_ACCESSPERMISSION_ACCESSIBLE_SECURED         = 0x02,
    SRFID_ACCESSPERMISSION_ALWAYS_NOT_ACCESSIBLE      = 0x03,
} SRFID_ACCESSPERMISSION;
```

SRFID_BEEPERCONFIG

Description

Used to represent a beeper configuration.

```
typedef enum {
    SRFID_BEEPERCONFIG_HIGH                           = 0x00,
    SRFID_BEEPERCONFIG_MEDIUM                         = 0x01,
    SRFID_BEEPERCONFIG_LOW                            = 0x02,
    SRFID_BEEPERCONFIG_QUIET                          = 0x03,
} SRFID_BEEPERCONFIG;
```

SRFID_TRIGGEREVENT

Used to represent an event related to a hand-held trigger.

```
typedef enum {
    SRFID_TRIGGEREVENT_PRESSED                        = 0x00,
    SRFID_TRIGGEREVENT_RELEASED                       = 0x01,
} SRFID_TRIGGEREVENT;
```

SRFID_HOPPINGCONFIG

Description

Used to represent an event related to a hand-held trigger.

```
typedef enum {
    SRFID_HOPPINGCONFIG_DEFAULT                        = 0x00,
    SRFID_HOPPINGCONFIG_ENABLED                        = 0x01,
    SRFID_HOPPINGCONFIG_DISABLED                       = 0x02,
} SRFID_HOPPINGCONFIG;
```

srfidReaderInfo

Description

Used to represent a specific RFID reader.

```
@interface srfidReaderInfo : NSObject
{
    int m_ReaderID;
    int m_ConnectionType;
    BOOL m_Active;
    NSString *m_ReaderName;
    int m_ReaderModel;
}

- (id)init;
- (void)dealloc;

- (void)setReaderID:(int)readerID;
- (void)setConnectionType:(int)connectionType;
- (void)setActive:(BOOL)active;
- (void)setReaderName:(NSString*)readerName;
- (void)setReaderModel:(int)readerModel;

- (int)getReaderID;
- (int)getConnectionType;
- (BOOL)isActive;
- (NSString*)getReaderName;
- (int)getReaderModel;

@end
```

Table 3-24 *srfidReaderInfo Variable Name Descriptions*

Variable Name	Description
m_ReaderID	Unique identifier of a specific RFID reader assigned by SDK.
m_ConnectionType	Communication mode of a specific RFID reader.
m_Active	State of a specific RFID reader (i.e., active RFID reader is an RFID reader with which a communication session was already established).
m_ReaderName	Name of a specific RFID reader.
m_ReaderModel	Model code of a specific RFID reader.

srfidISdkApi

Description

Objective C protocol which defines SDK API functions.

```
@protocol srfidISdkApi <NSObject>

- (NSString*) srfidGetSdkVersion;

- (SRFID_RESULT) srfidSetDelegate:(id<srfidISdkApiDelegate>)delegate;
- (SRFID_RESULT) srfidSubsribeForEvents:(int)sdkEventsMask;
- (SRFID_RESULT) srfidUnsubsribeForEvents:(int)sdkEventsMask;

- (SRFID_RESULT) srfidSetOperationalMode:(int)operationalMode;

- (SRFID_RESULT) srfidGetAvailableReadersList: (NSMutableArray**)availableReadersList;
- (SRFID_RESULT) srfidGetActiveReadersList:(NSMutableArray**)activeReadersList;

- (SRFID_RESULT) srfidEstablishCommunicationSession:(int)readerID;
- (SRFID_RESULT) srfidTerminateCommunicationSession:(int)readerID;
- (SRFID_RESULT) srfidEstablishAsciiConnection:(int)readerID
aPassword:(NSString*)password;

- (SRFID_RESULT) srfidEnableAvailableReadersDetection:(BOOL)enable;
- (SRFID_RESULT) srfidEnableAutomaticSessionReestablishment:(BOOL)enable;

- (SRFID_RESULT) srfidStartRapidRead:(int)readerID aReportConfig:
(srfidReportConfig*)reportConfig aAccessConfig:(srfidAccessConfig*)accessConfig
aStatusMessage:(NSString**)statusMessage;
- (SRFID_RESULT) srfidStopRapidRead:(int)readerID aStatusMessage:
(NSString**)statusMessage;

- (SRFID_RESULT) srfidStartInventory:(int)readerID aMemoryBank:
(SRFID_MEMORYBANK)memoryBankId aReportConfig:(srfidReportConfig*)reportConfig
aAccessConfig:(srfidAccessConfig*)accessConfig aStatusMessage: (NSString**)statusMessage;
- (SRFID_RESULT) srfidStopInventory:(int)readerID aStatusMessage:
(NSString**)statusMessage;

- (SRFID_RESULT) srfidGetSupportedLinkProfiles:(int)readerID
aLinkProfilesList:(NSMutableArray**)linkProfilesList aStatusMessage:
(NSString**)statusMessage;
- (SRFID_RESULT) srfidGetAntennaConfiguration:(int)readerID
aAntennaConfiguration:(srfidAntennaConfiguration**)antennaConfiguration
aStatusMessage:(NSString**)statusMessage;
- (SRFID_RESULT) srfidSetAntennaConfiguration:(int)readerID
aAntennaConfiguration:(srfidAntennaConfiguration*)antennaConfiguration
aStatusMessage:(NSString**)statusMessage;

- (SRFID_RESULT) srfidGetDpoConfiguration:(int)readerID
aDpoConfiguration:(srfidDynamicPowerConfig**)dpoConfiguration
aStatusMessage:(NSString**)statusMessage;

- (SRFID_RESULT) srfidSetDpoConfiguration:(int)readerID
aDpoConfiguration:(srfidDynamicPowerConfig*)dpoConfiguration
aStatusMessage:(NSString**)statusMessage;

- (SRFID_RESULT) srfidGetSingulationConfiguration:(int)readerID
aSingulationConfig:(srfidSingulationConfig**)singulationConfig aStatusMessage:
(NSString**)statusMessage;
```

(continued on next page)

```
- (SRFID_RESULT) srfidSetSingulationConfiguration:(int)readerID
aSingulationConfig:(srfidSingulationConfig*)singulationConfig aStatusMessage:
(NSString**)statusMessage;

- (SRFID_RESULT) srfidGetTagReportConfiguration:(int)readerID
aTagReportConfig:(srfidTagReportConfig**)reportConfig aStatusMessage:
(NSString**)statusMessage;
- (SRFID_RESULT) srfidSetTagReportConfiguration:(int)readerID
aTagReportConfig:(srfidTagReportConfig*)reportConfig aStatusMessage:
(NSString**)statusMessage

- (SRFID_RESULT) srfidGetReaderVersionInfo:(int)readerID
aReaderVersionInfo:(srfidReaderVersionInfo**)readerVersionInfo
aStatusMessage:(NSString**)statusMessage;
- (SRFID_RESULT) srfidGetReaderCapabilitiesInfo:(int)readerID
aReaderCapabilitiesInfo:(srfidReaderCapabilitiesInfo**)readerCapabilitiesInfo
aStatusMessage:(NSString**)statusMessage;

- (SRFID_RESULT) srfidGetStartTriggerConfiguration:(int)readerID
aStartTriggeConfig:(srfidStartTriggerConfig**)triggerConfig
aStatusMessage:(NSString**)statusMessage;
- (SRFID_RESULT) srfidSetStartTriggerConfiguration:(int)readerID
aStartTriggeConfig:(srfidStartTriggerConfig*)triggerConfig
aStatusMessage:(NSString**)statusMessage

- (SRFID_RESULT) srfidGetStopTriggerConfiguration:(int)readerID
aStopTriggeConfig:(srfidStopTriggerConfig**)triggerConfig
aStatusMessage:(NSString**)statusMessage;
- (SRFID_RESULT) srfidSetStopTriggerConfiguration:(int)readerID
aStopTriggeConfig:(srfidStopTriggerConfig*)triggerConfig
aStatusMessage:(NSString**)statusMessage

- (SRFID_RESULT) srfidGetSupportedRegions:(int)readerID
aSupportedRegions:(NSMutableArray**)supportedRegionsList
aStatusMessage:(NSString**)statusMessage;
- (SRFID_RESULT) srfidGetRegionInfo:(int)readerID aRegionCode:(NSString*)regionCode
aSupportedChannels:(NSMutableArray**)supportedChannelsList
aHoppingConfigurable:(BOOL*)hoppingConfigurable aStatusMessage:(NSString**)statusMessage;

- (SRFID_RESULT) srfidGetRegulatoryConfig:(int)readerID
aRegulatoryConfig:(srfidRegulatoryConfig**)regulatoryConfig
aStatusMessage:(NSString**)statusMessage;
- (SRFID_RESULT) srfidSetRegulatoryConfig:(int)readerID
aRegulatoryConfig:(srfidRegulatoryConfig*)regulatoryConfig
aStatusMessage:(NSString**)statusMessage;

- (SRFID_RESULT) srfidGetBeeperConfig:(int)readerID
aBeeperConfig:(SRFID_BEEPERCONFIG*)beeperConfig aStatusMessage:(NSString**)statusMessage;
- (SRFID_RESULT) srfidSetBeeperConfig:(int)readerID
aBeeperConfig:(SRFID_BEEPERCONFIG)beeperConfig aStatusMessage:(NSString**)statusMessage;

- (SRFID_RESULT) srfidGetPreFilters:(int)readerID
aPreFilters:(NSMutableArray**)filtersList aStatusMessage:(NSString**)statusMessage;
- (SRFID_RESULT) srfidSetPreFilters:(int)readerID aPreFilters:(NSMutableArray*)filtersList
aStatusMessage:(NSString**)statusMessage;
```

(continued on next page)

```
- (SRFID_RESULT) srfidStartTagLocationing:(int)readerID aTagEpcId:(NSString*)epcID
aStatusMessage:(NSString**)statusMessage;
- (SRFID_RESULT) srfidStopTagLocationing:(int)readerID
aStatusMessage:(NSString**)statusMessage

- (SRFID_RESULT) srfidSaveReaderConfiguration:(int)readerID
aSaveCustomDefaults:(BOOL)saveCustomDefaults aStatusMessage:(NSString**)statusMessage;
- (SRFID_RESULT) srfidRestoreReaderConfiguration:(int)readerID
aRestoreFactoryDefaults:(BOOL)restoreFactoryDefaults
aStatusMessage:(NSString**)statusMessage;

- (SRFID_RESULT) srfidReadTag:(int)readerID aTagID:(NSString*)tagID
aAccessTagData:(srfidTagData**)accessTagData aMemoryBank:(SRFID_MEMORYBANK)memoryBankID
aOffset:(short)offset aLength:(short)length aPassword:(long)password
aStatusMessage:(NSString**)statusMessage;
- (SRFID_RESULT) srfidWriteTag:(int)readerID aTagID:(NSString*)tagID
aAccessTagData:(srfidTagData**)accessTagData aMemoryBank:(SRFID_MEMORYBANK)memoryBankID
aOffset:(short)offset aData:(NSString*)data aPassword:(long)password
aDoBlockWrite:(BOOL)blockWrite aStatusMessage:(NSString**)statusMessage;
- (SRFID_RESULT) srfidKillTag:(int)readerID aTagID:(NSString*)tagID
aAccessTagData:(srfidTagData**)accessTagData aPassword:(long)password
aStatusMessage:(NSString**)statusMessage;
- (SRFID_RESULT) srfidLockTag:(int)readerID aTagID:(NSString*)tagID
aAccessTagData:(srfidTagData**)accessTagData aMemoryBank:(SRFID_MEMORYBANK)memoryBankID
aAccessPermissions:(SRFID_ACCESSPERMISSION)accessPermissions aPassword:(long)password
aStatusMessage:(NSString**)statusMessage;"

- (SRFID_RESULT) srfidBlockErase:(int)readerID aTagID:(NSString *)tagID
aAccessTagData:(srfidTagData**)accessTagData aMemoryBank:(SRFID_MEMORYBANK)memoryBankID
aOffset:(short)offset aLength:(short)length aPassword:(long)password
aStatusMessage:(NSString**)statusMessage;
- (SRFID_RESULT) srfidBlockPermaLock:(int)readerID aTagID:(NSString *)tagID
aAccessTagData:(srfidTagData**)accessTagData aMemoryBank:(SRFID_MEMORYBANK)memoryBankID
aDoLock:(BOOL)doLock aBlockPtr:(short)blockPtr aBlockRange:(short)blockRange
aBlockMask:(NSString *)blockMask aPassword:(long)password aStatusMessage:(NSString
**)statusMessage;

- (SRFID_RESULT) srfidReadTag:(int)readerID
aAccessCriteria:(srfidAccessCriteria*)accessCriteria
aAccessTagData:(srfidTagData**)accessTagData aMemoryBank:(SRFID_MEMORYBANK)memoryBankID
aOffset:(short)offset aLength:(short)length aPassword:(long)password
aStatusMessage:(NSString**)statusMessage;
- (SRFID_RESULT) srfidWriteTag:(int)readerID
aAccessCriteria:(srfidAccessCriteria*)accessCriteria
aAccessTagData:(srfidTagData**)accessTagData aMemoryBank:(SRFID_MEMORYBANK)memoryBankID
aOffset:(short)offset aData:(NSString*)data aPassword:(long)password
aDoBlockWrite:(BOOL)blockWrite aStatusMessage:(NSString**)statusMessage;
- (SRFID_RESULT) srfidKillTag:(int)readerID
aAccessCriteria:(srfidAccessCriteria*)accessCriteria
aAccessTagData:(srfidTagData**)accessTagData aPassword:(long)password
aStatusMessage:(NSString**)statusMessage;
- (SRFID_RESULT) srfidLockTag:(int)readerID
aAccessCriteria:(srfidAccessCriteria*)accessCriteria
aAccessTagData:(srfidTagData**)accessTagData aMemoryBank:(SRFID_MEMORYBANK)memoryBankID
aAccessPermissions:(SRFID_ACCESSPERMISSION)accessPermissions aPassword:(long)password
aStatusMessage:(NSString**)statusMessage;
```

(continued on next page)

```
- (SRFID_RESULT) srfidBlockErase:(int)readerID
aAccessCriteria:(srfidAccessCriteria*)accessCriteria
aAccessTagData:(srfidTagData**)accessTagData aMemoryBank:(SRFID_MEMORYBANK)memoryBankID
aOffset:(short)offset aLength:(short)length aPassword:(long)password
aStatusMessage:(NSString**)statusMessage;
- (SRFID_RESULT) srfidBlockPermaLock:(int)readerID
aAccessCriteria:(srfidAccessCriteria*)accessCriteria
aAccessTagData:(srfidTagData**)accessTagData aMemoryBank:(SRFID_MEMORYBANK)memoryBankID
aDoLock:(BOOL)doLock aBlockPtr:(short)blockPtr aBlockRange:(short)blockRange
aBlockMask:(NSString *)blockMask aPassword:(long)password aStatusMessage:(NSString
**)statusMessage;

aAccessTagData:(srfidTagData**)accessTagData aMemoryBank:(SRFID_MEMORYBANK)memoryBankID
aAccessPermissions:(SRFID_ACCESSPERMISSION)accessPermissions aPassword:(long)password
aStatusMessage:(NSString**)statusMessage;

-    (SRFID_RESULT) srfidRequestBatteryStatus:(int)readerID;
-    (SRFID_RESULT) srfidGetBatchModeConfig:(int)readerID
aBatchModeConfig:(SRFID_BATCHMODECONFIG*)batchModeConfig
aStatusMessage:(NSString**)statusMessage;
-    (SRFID_RESULT) srfidSetBatchModeConfig:(int)readerID
aBatchModeConfig:(SRFID_BATCHMODECONFIG)batchModeConfig
aStatusMessage:(NSString**)statusMessage;
-    (SRFID_RESULT) srfidgetTags:(int)readerID aStatusMessage:(NSString**)statusMessage;
-    (SRFID_RESULT) srfidGetReaderBatchModeEvent:(int) readerID;
-    (SRFID_RESULT) srfidPurgeTags:(int)readerID aStatusMessage:(NSString**)statusMessage;
-    (SRFID_RESULT) srfidSetAttribute:(int)readerId attributeNumber:(int)attrNum
attributeValue:(int)attrVal attributeType:(NSString*)attrType
aStatusMessage:(NSString**)statusMessage;
-    (SRFID_RESULT) srfidGetAttribute:(int)readerID aAttrNum:(int)attrNum
aAttrInfo:(srfidAttribute**)attrInfo aStatusMessage:(NSString**)statusMessage;
-    (SRFID_RESULT) srfidSetAccessCommandOperationWaitTimeout:(int)readerID
aTimeoutMs:(int)timeoutMs;
-    (SRFID_RESULT) srfidLocateReader:(int)readerID doEnabled:(BOOL)doEnabled
aStatusMessage:(NSString**)statusMessage;

@end
```

See *Functions on page 3-41* for descriptions of API functions.

srfidSdkFactory

Description

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

```
@interface srfidSdkFactory : NSObject

+(id<srfidISdkApi>)createRfidSdkApiInstance;

@end
```

+(id<srfidISdkApi>)createRfidSdkApiInstance

Class method, returns a single shared instance of an object which conforms to srfidISdkApi protocol and should be used to perform specific API calls.

srfidISdkApiDelegate

Description

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

```
@protocol srfidISdkApiDelegate <NSObject>

- (void)srfidEventReaderAppeared:(srfidReaderInfo*)availableReader;
- (void)srfidEventReaderDisappeared:(int)readerID;
- (void)srfidEventCommunicationSessionEstablished:(srfidReaderInfo*)activeReader;
- (void)srfidEventCommunicationSessionTerminated:(int)readerID;
- (void)srfidEventReadNotify:(int)readerID aTagData:(srfidTagData*)tagData;
- (void)srfidEventStatusNotify:(int)readerID aEvent:(SRFID_EVENT_STATUS)event
aNotification:(id)notificationData;
- (void)srfidEventProximityNotify:(int)readerID aProximityPercent:(int)proximityPercent;
- (void)srfidEventTriggerNotify:(int)readerID
aTriggerEvent:(SRFID_TRIGGEREVENT)triggerEvent;
- (void)srfidEventBatteryNotity:(int)readerID
aBatteryEvent:(srfidBatteryEvent*)batteryEvent;

@end
```

See *Notifications on page 3-108* for descriptions of SDK callbacks.

srfidTagData

Description

Used to represent a single item of tag related information received from a specific RFID reader.

```
@interface srfidTagData : NSObject
{
    NSMutableString *m_TagId;
    long m_FirstSeenTime;
    long m_LastSeenTime;
    int m_PC;
    short m_PeakRSSI;
    short m_PhaseInfo;
    short m_ChannelIndex;
    short m_TagSeenCount;
    srfidAccessOperationCode *m_OpCode;
    BOOL m_OperationSucceed;
    NSMutableString *m_OperationStatus;
    SRFID_MEMORYBANK m_MemoryBank;
    NSMutableString *m_MemoryBankData;
    NSMutableString *m_PermaLockData;
    int m_ModifiedWordCount;

}

- (NSString*)getTagId;
- (void)setTagId:(NSString*)val;
- (long)getFirstSeenTime;
- (void)setFirstSeenTime:(long)val;
- (long)getLastSeenTime;
- (void)setLastSeenTime:(long)val;
- (int)getPC;
- (void)setPC:(int)val;
- (short)getPeakRSSI;
- (void)setPeakRSSI:(short)val;
- (short)getChannelIndex;
- (void)setChannelIndex:(short)val;
- (short)getPhaseInfo;
- (void)setPhaseInfo:(short)val;
- (short)getTagSeenCount;
- (void)setTagSeenCount:(short)val;
- (srfidAccessOperationCode*)getOpCode;
- (void)setOpCode:(srfidAccessOperationCode*)val;
- (BOOL)getOperationSucceed;
- (void)setOperationSucceed:(BOOL)val;
- (NSString*)getOperationStatus;
- (void)setOperationStatus:(NSString*)val;
- (SRFID_MEMORYBANK)getMemoryBank;
- (void)setMemoryBank:(SRFID_MEMORYBANK)val;
- (NSString*)getMemoryBankData;
- (void)setMemoryBankData:(NSString*)val;
- (NSString*)getPermaLockData;
- (void)setPermaLockData:(NSString *)val;
- (int)getModifiedWordCount;
- (void)setModifiedWordCount:(int)val;

@end
```

Table 3-25 *srfidTagData Variable Descriptions*

Variable Name	Description
m_TagId	Identifier of a received RFID tag.
m_FirstSeenTime	First seen time information related to a received RFID tag.
m_LastSeenTime	Last seen time information related to a received RFID tag.
m_PC	PC information related to a received RFID tag.
m_PeakRSSI	RSSI information related to a received RFID tag.
m_PhaseInfo	Phase information related to a received RFID tag.
m_ChannelIndex	Channel information related to a received RFID tag.
m_TagSeenCount	Tag seen count information related to a received RFID tag.
m_OpCode	A specific access operation related to a received RFID tag.
m_OperationSucceed	Boolean value indicating whether an access operation related to a received RFID tag has been performed successfully.
m_OperationStatus	String value indicating an occurred error if an access operation related to a received RFID tag has failed.
m_MemoryBank	Identifier of memory bank that was used to perform a specific access operation.
m_MemoryBankData	Data received from a specific memory bank of an RFID tag.
m_PermaLockData	Perma lock data received from a particular memory bank of an RFID tag.
m_ModifiedWordCount	Number of words modified after performing Access operations (write, blockerase etc.).

srfidTagFilter

Description

Used to represent a tag filter that is used for selecting a tag to access.

```
@interface srfidTagFilter : NSObject
{
SRFID_MEMORYBANK m_Filter_MaskBank;
short m_Filter_MaskStartPos;
NSMutableString *m_Filter_Data;
NSMutableString *m_Filter_Mask;
short m_Filter_MatchLength;
BOOL m_Filter_DoMatch;
}
- (SRFID_MEMORYBANK)getFilterMaskBank;
- (void)setFilterMaskBank:(SRFID_MEMORYBANK)val;
- (short)getFilterMaskStartPos;
- (void)setFilterMaskStartPos:(short)val;
- (NSString*)getFilterData;
- (void)setFilterData:(NSString*)val;
- (NSString*)getFilterMask;
- (void)setFilterMask:(NSString*)val;
- (short)getFilterMatchLength;
- (void)setFilterMatchLength:(short)val;
- (BOOL)getFilterDoMatch;
- (void)setFilterDoMatch:(BOOL)val;
@end
```

Table 3-26 *srfidTagFilter Variable Descriptions*

Variable Name	Description
m_Filter_MaskBank	Specifies the memory bank to access.
m_Filter_MaskStartPos	Specifies start bit position from beginning of memory bank from where match pattern is checked.
m_Filter_Data	Specifies the data pattern to filter.
m_Filter_Mask	Specifies the bit mask for bits to check in pattern.
m_Filter_MatchLength	Specifies the number of bits from start of match pattern to be used for matching.
m_Filter_DoMatch	Specifies to operate on matching tag. Set to FALSE to specify to operate on non-matching tag.

srfidAccessOperationCode

Description

Used to represent information (operation code and string representation) regarding one of supported access operations.

```
@interface srfidAccessOperationCode : NSObject
{
    NSString *m_Name;
    SRFID_ACCESSOPERATIONCODE m_Ordinal;
}

- (id)initAccessOperationCode:(NSString*)name aOrdinal:(SRFID_ACCESSOPERATIONCODE)ordinal;

+ (srfidAccessOperationCode*)getAccessOperationCodeValue:
(SRFID_ACCESSOPERATIONCODE)value;
+ (srfidAccessOperationCode*)getAccessOperationCodeValueFromString: (NSString*)str;

- (NSString*)getName;
- (SRFID_ACCESSOPERATIONCODE)getOrdinal;

@end
```

Table 3-27 *srfidAccessOperationCode Variable Descriptions*

Variable Name	Description
m_Name	String representation of an access operation code.
m_Ordinal	Access operation code. + (srfidAccessOperationCode*)getAccessOperationCodeValue: (SRFID_ACCESSOPERATIONCODE)value Class method, returns a srfidAccessOperationCode object in accordance with received access operation code. + (srfidAccessOperationCode*)getAccessOperationCodeValueFromString:(NSString*)str Class method, returns a srfidAccessOperationCode object in accordance with received string representation of an access operation code.

srfidAccessConfig

Description

Used to represent access parameters for some of API calls.

```
@interface srfidAccessConfig : NSObject
{
    BOOL m_DoSelect;
    short m_Power;
}

- (BOOL)getDoSelect;
- (void)setDoSelect:(BOOL)val;
- (short)getPower;
- (void)setPower:(short)val;

@end
```

Table 3-28 *srfidAccessConfig Variable Descriptions*

Variable Name	Description
m_DoSelect	Specifies whether select records (prefilters) is applied.
m_Power	Specifies selected output power in 0.1 dBm units.

srfidAccessCriteria

Description

Used to represent access criteria for some of API calls. Each criteria can have two tag filters.

```
@interface srfidAccessConfig : NSObject
@property (atomic, retain) srfidTagFilter *tagFilter1;
@property (atomic, retain) srfidTagFilter *tagFilter2;
@end
```

Table 3-29 *srfidReportConfig Variable Descriptions*

Variable Name	Description
tagFilter1	Specifies tag filter #1 information for access criteria.
tagFilter2	Specifies tag filter #2 information for access criteria.

srfidReportConfig

Description

Used to represent report parameters for some of API calls.

```
@interface srfidReportConfig : NSObject
{
    BOOL m_IncFirstSeenTime;
    BOOL m_IncLastSeenTime;
    BOOL m_IncPC;
    BOOL m_IncRSSI;
    BOOL m_IncPhase;
    BOOL m_IncChannelIndex;
    BOOL m_IncTagSeenCount;
}

- (BOOL)getIncFirstSeenTime;
- (void)setIncFirstSeenTime:(BOOL)val;
- (BOOL)getIncLastSeenTime;
- (void)setIncLastSeenTime:(BOOL)val;
- (BOOL)getIncPC;
- (void)setIncPC:(BOOL)val;
- (BOOL)getIncRSSI;
- (void)setIncRSSI:(BOOL)val;
- (BOOL)getIncPhase;
- (void)setIncPhase:(BOOL)val;
- (BOOL)getIncChannelIndex;
- (void)setIncChannelIndex:(BOOL)val;
- (BOOL)getIncTagSeenCount;
- (void)setIncTagSeenCount:(BOOL)val;

@end
```

Table 3-30 *srfidReportConfig Variable Descriptions*

Variable Name	Description
m_IncFirstSeenTime	Specifies whether first seen time information related to a specific RFID tag is reported.
m_ IncLastSeenTime	Specifies whether last seen time information related to a specific RFID tag is reported.
m_ IncPC	Specifies whether PC information related to a specific RFID tag is reported.
m_ IncRSSI	Specifies whether RSSI information related to a specific RFID tag is reported.
m_ IncPhase	Specifies whether phase information related to a specific RFID tag is reported.
m_ IncChannelIndex	Specifies whether channel information related to a specific RFID tag is reported.
m_ IncTagSeenCount	Specifies whether seen count information related to a specific RFID tag is reported.

srfidSingulationConfig

Description

Used to represent singulation parameters of a specific RFID reader.

```
@interface srfidSingulationConfig : NSObject
{
    SRFID_SLFLAG m_SLFlag;
    SRFID_SESSION m_Session;
    SRFID_INVENTORYSTATE m_State;
    int m_TagPopulation;
}

- (SRFID_SLFLAG)getSLFlag;
- (void)setSlFlag:(SRFID_SLFLAG)val;
- (SRFID_SESSION)getSession;
- (void)setSession:(SRFID_SESSION)val;
- (SRFID_INVENTORYSTATE)getInventoryState;
- (void)setInventoryState:(SRFID_INVENTORYSTATE)val;
- (int)getTagPopulation;
- (void)setTagPopulation:(int)val;

@end
```

Table 3-31 *srfidSingulationConfig Variable Descriptions*

Variable Name	Description
m_SLFlag	Value of selection (SL flag) parameter of singulation configuration.
m_Session	Value of session parameter of singulation configuration.
m_InventoryState	Value of target (inventory state) parameter of singulation configuration.
m_TagPopulation	Value of tag population parameter of singulation configuration.

srfidLinkProfile

Description

Used to represent link profile (RF mode) related information.

```
@interface srfidLinkProfile : NSObject
{
    int m_RFModeIndex;
    SRFID_DIVIDERATIO m_DivideRatio;
    int m_BDR;
    SRFID_MODULATION m_Modulation;
    SRFID_FORWARDLINKMODULATION m_FLModulation;
    int m_PIE;
    int m_MinTari;
    int m_MaxTari;
    int m_StepTari;
    SRFID_SPECTRALMASKINDICATOR m_SpectralMaskIndicator;
    BOOL m_EPCHAGTCConformance;
}
```

(continued on next page)

```
- (int)getRFModeIndex;
- (void)setRFModeIndex:(int)val;
- (SRFID_DIVIDERATIO)getDivideRatio;
- (void)setDivideRatio:(SRFID_DIVIDERATIO)val;
- (int)getBDR;
- (void)setBDR:(int)val;
- (SRFID_MODULATION)getModulation;
- (void)setModulation:(SRFID_MODULATION)val;
- (SRFID_FORWARDLINKMODULATION)getFLModulation;
- (void)setFLModulation:(SRFID_FORWARDLINKMODULATION)val;
- (int)getPIE;
- (void)setPIE:(int)val;
- (int)getMinTari;
- (void)setMinTari:(int)val;
- (int)getMaxTari;
- (void)setMaxTari:(int)val;
- (int)getStepTari;
- (void)setStepTari:(int)val;
- (SRFID_SPECTRALMASKINDICATOR)getSpectralMaskIndicator;
- (void)setSpectralMaskIndicator:(SRFID_SPECTRALMASKINDICATOR)val;
- (BOOL)getEPCHAGTCConformance;
- (void)setEPCHAGTCConformance:(BOOL)val;
- (NSString*)getDivideRatioString;
- (NSString*)getModulationString;
- (NSString*)getForwardLinkModulationString;
- (NSString*)getSpectralMaskIndicatorString;

@end
```

Table 3-32 *srfidLinkProfile Variable Descriptions*

Variable Name	Description
m_RFModeIndex	Index of a specific link profile.
m_DivideRatio	Divide ratio of a specific link profile.
m_BDR	BDR of a specific link profile.
m_Modulation	Modulation of a specific link profile.
m_FLModulation	Forward link modulation of a specific link profile.
m_PIE	PIE of a specific link profile.
m_MinTari	Minimal tari value of a specific link profile.
m_MaxTari	Maximal tari value of a specific link profile.
m_StepTari	Step tari value of a specific link profile.
m_Spectral MaskIndicator	Spectral mask indicator of a specific link profile.
m_EPCHAGTCConformance	EPCHAGT&CConformance value of a specific link profile.

srfidAntennaConfiguration

Description

Used to represent antenna related configuration of a specific RFID reader.

```
@interface srfidAntennaConfiguration : NSObject
{
    short m_Power;
    short m_LinkProfileIdx;
    int m_Tari;
    BOOL m_DoSelect;
}

- (short)getPower;
- (void)setPower:(short)val;
- (short)getLinkProfileIdx;
- (void)setLinkProfileIdx:(short)val;
- (int)getTari;
- (void)setTari:(int)val;
- (BOOL)getDoSelect;
- (void)setDoSelect:(BOOL)val;

@end
```

Table 3-33 *srfidAntennaConfiguration Variable Descriptions*

Variable Name	Description
m_Power	Selected output power in 0.1 dBm units.
m_LinkProfileIdx	Index of selected link profile (RF mode).
m_Tari	Selected tari value.
m_DoSelect	Specifies whether select records (pre-filters) is applied for operations.

srfidDynamicPowerConfig

Description

Used to represent dynamic power optimization related configuration of a specific RFID reader.

```
@interface srfidDynamicPowerConfig : NSObject
{
    BOOL m_DynamicPowerOptimizationEnabled;
}

- (BOOL)getDynamicPowerOptimizationEnabled;
- (void)setDynamicPowerOptimizationEnabled:(BOOL)val;

@end
```

Table 3-34 *srfidDynamicPowerConfig Variable Descriptions*

Variable Name	Description
m_DynamicPowerOptimizationEnabled	Specifies whether the dynamic power optimization feature shall be enabled.

srfidTagReportConfig

Description

Used to configure fields that are reported in a response to operation.

```
@interface srfidTagReportConfig : NSObject
{
    BOOL m_IncFirstSeenTime;
    BOOL m_IncLastSeenTime;
    BOOL m_IncPC;
    BOOL m_IncRSSI;
    BOOL m_IncPhase;
    BOOL m_IncChannelIdx;
    BOOL m_IncTagSeenCount;
}

- (BOOL)getIncFirstSeenTime;
- (void)setIncFirstSeenTime:(BOOL)val;
- (BOOL)getIncLastSeenTime;
- (void)setIncLastSeenTime:(BOOL)val;
- (BOOL)getIncPC;
- (void)setIncPC:(BOOL)val;
- (BOOL)getIncRSSI;
- (void)setIncRSSI:(BOOL)val;
- (BOOL)getIncPhase;
- (void)setIncPhase:(BOOL)val;
- (BOOL)getIncChannelIdx;
- (void)setIncChannelIdx:(BOOL)val;
- (BOOL)getIncTagSeenCount;
- (void)setIncTagSeenCount:(BOOL)val;

@end
```

Table 3-35 *srfidTagReportConfig Variable Descriptions*

Variable Name	Description
m_IncFirstSeenTime	Specifies whether first seen time information related to a specific RFID tag is reported.
m_ IncLastSeenTime	Specifies whether last seen time information related to a specific RFID tag is reported.
m_ IncPC	Specifies whether PC information related to a specific RFID tag is reported.
m_ IncRSSI	Specifies whether RSSI information related to a specific RFID tag is reported.
m_ IncPhase	Specifies whether phase information related to a specific RFID tag is reported.
m_ IncChannelIndex	Specifies whether channel information related to a specific RFID tag is reported.
m_ IncTagSeenCount	Specifies whether seen count information related to a specific RFID tag is reported.

srfidStartTriggerConfig

Description

Used to represent start trigger parameters of a specific RFID reader.

```
@interface srfidStartTriggerConfig : NSObject
{
    BOOL m_StartOnHandheldTrigger;
    SRFID_TRIGGERTYPE m_TriggerType;
    int m_StartDelay;
    BOOL m_RepeatMonitoring;
}

- (BOOL)getStartOnHandheldTrigger;
- (void)setStartOnHandheldTrigger:(BOOL)val;
- (SRFID_TRIGGERTYPE)getTriggerType;
- (void)setTriggerType:(SRFID_TRIGGERTYPE)val;
- (int)getStartDelay;
- (void)setStartDelay:(int)val;
- (BOOL)getRepeatMonitoring;
- (void)setRepeatMonitoring:(BOOL)val;

@end
```

Table 3-36 *srfidStartTriggerConfig Variable Descriptions*

Variable Name	Description
m_StartOnHandheldTrigger	Specifies whether start of operation on physical trigger shall be enabled.
m_TriggerType	Specifies trigger type, used if m_StartOnHandheldTrigger is set.
m_StartDelay	Specifies a value of delay in milliseconds for start of operation.
m_RepeatMonitoring	Specifies whether a monitoring for start trigger shall be repeated after stop of operation.

srfidStopTriggerConfig

Description

Used to represent stop trigger parameters of a specific RFID reader.

```
@interface srfidStopTriggerConfig : NSObject
{
    BOOL m_StopOnHandheldTrigger;
    SRFID_TRIGGERTYPE m_TriggerType;
    BOOL m_StopOnTagCount;
    int m_StopTagCount;
    BOOL m_StopOnTimeout;
    int m_StopTimeout;
    BOOL m_StopOnInventoryCount;
    int m_StopInventoryCount;
    BOOL m_StopOnAccessCount;
    int m_StopAccessCount;
}

- (BOOL)getStopOnHandheldTrigger;
- (void)setStopOnHandheldTrigger:(BOOL)val;
- (SRFID_TRIGGERTYPE)getTriggerType;
- (void)setTriggerType:(SRFID_TRIGGERTYPE)val;
- (BOOL)getStopOnTagCount;
- (void)setStopOnTagCount:(BOOL)val;
- (int)getStopTagCount;
- (void)setStopTagCount:(int)val;
- (BOOL)getStopOnTimeout;
- (void)setStopOnTimeout:(BOOL)val;
- (int)getStopTimeout;
- (void)setStopTimout:(int)val;
- (BOOL)getStopOnInventoryCount;
- (void)setStopOnInventoryCount:(BOOL)val;
- (int)getStopInventoryCount;
- (void)setStopInventoryCount:(int)val;
- (BOOL)getStopOnAccessCount;
- (void)setStopOnAccessCount:(BOOL)val;
- (int)getStopAccessCount;
- (void)setStopAccessCount:(int)val;

@end
```

Table 3-37 *srfidStopTriggerConfig Variable Descriptions*

Variable Name	Description
m_StopOnHandheldTrigger	Specifies whether stop of operation on physical trigger shall be enabled.
m_TriggerType	Specifies trigger type, used if m_StopOnHandheldTrigger is set.
m_StopOnTagCount	Specifies whether an on-going operation shall be stopped after the number of tags has been inventoried.
m_StopTagCount	Specifies the number of tags to be inventoried before stop of on-going operation, used if m_StopOnTagCount is set.
m_StopOnTimeout	Specifies whether stop on operation on timeout shall be enabled.
m_StopTimeout	Specifies the value of timeout in milliseconds, used if m_StopOnTimeout is set.

Table 3-37 *srfidStopTriggerConfig Variable Descriptions (Continued)*

Variable Name	Description
m_StopOnInventoryCount	Specifies whether stop on operation based on number of inventory rounds completed shall be enabled.
m_StopInventoryCount	Specifies the number of inventory rounds to be completed before stop of on-going operation, used if m_StopOnInventoryCount is set.
m_StopOnAccessCount	Specifies whether stop on operation based on number of access rounds completed shall be enabled.
m_StopAccessCount	Specifies the number of access rounds to be completed before stop of on-going operation, used if m_StopOnAccessCount is set.

srfidReaderVersionInfo

Description

Used to represent software version information regarding a specific RFID reader.

```
@interface srfidReaderVersionInfo : NSObject
{
    NSString *m_DeviceVersion;
    NSString *m_BluetoothVersion;
    NSString *m_NGEVersion;
    NSString *m_PL33;
}

- (NSString*)getDeviceVersion;
- (void)setDeviceVersion:(NSString*)val;
- (NSString*)getBluetoothVersion;
- (void)setBluetoothVersion:(NSString*)val;
- (NSString*)getNGEVersion;
- (void)setNGEVersion:(NSString*)val;
- (void)setPL33:(NSString*)val;
- (NSString*)getPL33;

@end
```

Table 3-38 *srfidReaderVersionInfo Variable Descriptions*

Variable Name	Description
m_DeviceVersion	Specifies the version of device software.
m_BluetoothVersion	Specifies the version of Bluetooth software.
m_NGEVersion	Specifies the version of NGE software.
m_PL33	Specifies the version of PL33.

srfidRegionInfo

Description

Used to represent detailed information regarding one of regions supported by a specific RFID reader.

```
@interface srfidRegionInfo : NSObject
{
    NSMutableString *m_RegionCode;
    NSMutableString *m_RegionName;
}

- (NSString*)getRegionCode;
- (void)setRegionCode:(NSString*)val;
- (NSString*)getRegionName;
- (void)setRegionName:(NSString*)val;

@end
```

Table 3-39 *srfidRegionInfo Variable Descriptions*

Variable Name	Description
m_RegionCode	Specifies the three letter code of a specific region.
m_RegionName	Specifies the name of a specific region.

srfidRegulatoryConfig

Used to represent regulatory parameters of a specific RFID reader.

```
@interface srfidRegulatoryConfig : NSObject
{
    NSMutableString *m_RegionCode;
    NSMutableArray *m_EnabledChannelsList;
    SRFID_HOPPINGCONFIG m_HoppingConfig;
}

- (NSString*)getRegionCode;
- (void)setRegionCode:(NSString*)val;
- (NSArray*)getEnabledChannelsList;
- (void)setEnabledChannelsList:(NSArray*)val;
- (void)addEnabledChannel:(int)channelId;
- (SRFID_HOPPINGCONFIG)getHoppingConfig;
- (void)setHoppingConfig:(SRFID_HOPPINGCONFIG)val;

@end
```

Table 3-40 *srfidRegulatoryConfig Variable Descriptions*

Variable Name	Description
m_RegionCode	Specifies the three letter region code.
m_EnabledChannels	Specifies the set of enabled channels (each channel is represented by NSString object).
m_HoppingConfig	Specifies whether hopping is enabled.

srfidPreFilter

Description

Used to represent one of prefilters (select records) configured on a specific RFID reader.

```
@interface srfidPreFilter : NSObject
{
    SRFID_SELECTTARGET m_Target;
    SRFID_SELECTACTION m_Action;
    SRFID_MEMORYBANK m_MemoryBank;
    int m_MaskStartPos;
    NSMutableString *m_MatchPattern;
}

- (SRFID_SELECTTARGET)getTarget;
- (void)setTarget:(SRFID_SELECTTARGET)val;
- (SRFID_SELECTACTION)getAction;
- (void)setAction:(SRFID_SELECTACTION)val;
- (SRFID_MEMORYBANK)getMemoryBank;
- (void)setMemoryBank:(SRFID_MEMORYBANK)val;
- (int)getMaskStartPos;
- (void)setMaskStartPos:(int)val;
- (NSString*)getMatchPattern;
- (void)setMatchPattern:(NSString*)val;


@end
```

Table 3-41 *srfidPreFilter Variable Descriptions*

Variable Name	Description
m_Target	Specifies the value of target parameter.
m_Action	Specifies the value of action parameter.
m_MemoryBank	Specifies a specific memory bank to be used for checking match pattern.
m_MaskStartPos	Specifies the start position in words from beginning of memory bank from where the match pattern is checked.
m_MatchPattern	Specifies the match pattern as hex string.

srfidReaderCapabilitiesInfo

Description

Used to represent detailed information regarding capabilities of a specific RFID reader.

```
@interface srfidReaderCapabilitiesInfo : NSObject
{
    NSString *m_SerialNumber;
    NSString *m_Model;
    NSString *m_Manufacturer;
    NSString *m_ManufacturingDate;
    NSString *m_ScannerName;
    NSString *m_AsciiVersion;
    int m_SelectFilterNum;
    int m_MinPower;
    int m_MaxPower;
    int m_PowerStep;
    NSString *m_AirProtocolVersion;
    NSString *m_BDAddress;
    int m_MaxAccessSequence;
}

- (NSString*)getSerialNumber;
- (void)setSerialNumber:(NSString*)val;
- (NSString*)getModel;
- (void)setModel:(NSString*)val;
- (NSString*)getManufacturer;
- (void)setManufacturer:(NSString*)val;
- (NSString*)getManufacturingDate;
- (void)setManufacturingDate:(NSString*)val;
- (NSString*)getScannerName;
- (void)setScannerName:(NSString*)val;
- (NSString*)getAsciiVersion;
- (void)setAsciiVersion:(NSString*)val;
- (NSString*)getAirProtocolVersion;
- (void)setAirProtocolVersion:(NSString*)val;
- (NSString*)getBDAddress;
- (void)setBDAddress:(NSString*)val;
- (int)getSelectFilterNum;
- (void)setSelectFilterNum:(int)val;
- (int)getMinPower;
- (void)setMinPower:(int)val;
- (int)getMaxPower;
- (void)setMaxPower:(int)val;
- (int)getPowerStep;
- (void)setPowerStep:(int)val;
- (int)getMaxAccessSequence;
- (void)setMaxAccessSequence:(int)val;

@end
```

Table 3-42 *srfidReaderCapabilitiesInfo Variable Descriptions*

Variable Name	Description
m_SerialNumber	Specifies the serial number of a specific RFID reader.
m_Model	Specifies the model of a specific RFID reader.
m_Manufacturer	Specifies the manufacturer of a specific RFID reader.

Table 3-42 *srfidReaderCapabilitiesInfo Variable Descriptions (Continued)*

Variable Name	Description
m_ManufacturingDate	Specifies the manufacturing date of a specific RFID reader.
m_ScannerName	Specifies the imager of a specific RFID reader.
m_AsciiVersion	Specifies the version of ASCII protocol supported by a specific RFID reader.
m_AirProtocolVersion	Specifies the version of air protocol supported by a specific RFID reader.
m_BDAddress	Specifies the Bluetooth address of a specific RFID reader.
m_SelectFilterNum	Specifies the number of select records (pre-filters) supported by a specific RFID reader.
m_MinPower	Specifies the minimal antenna power level in 0.1 dBm units supported by a specific RFID reader.
m_MaxPower	Specifies the maximal antenna power level in 0.1 dBm units supported by a specific RFID reader.
m_PowerStep	Specifies the step of antenna power level in 0.1 dBm units supported by a specific RFID reader.
m_MaxAccessSequence	Specifies the maximal number of operations to be combined in a sequence supported by a specific RFID reader.

srfidBatteryEvent

Description

Used to represent a specific battery information related notification received from a specific active RFID reader.

```
@interface srfidBatteryEvent : NSObject
{
    int m_PowerLevel;
    BOOL m_IsCharging;
    NSMutableString *m_EventCause;
}

- (int)getPowerLevel;
- (void)setPowerLevel:(int)val;
- (BOOL)getIsCharging;
- (void)setIsCharging:(BOOL)val;
- (NSString*)getEventCause;
- (void)setEventCause:(NSString*)val;

@end
```

Table 3-43 *srfidBatteryEvent Variable Descriptions*

Variable Name	Description
m_PowerLevel	Battery power level.
m_IsCharging	Whether or not batter is charging now.
m_EventCause	String representation of the cause of received notification.

srfidOperEndSummaryEvent

Description

Used to represent information related to Operation End Summary notification received from a specific active reader.

```
@interface srfidOperEndSummaryEvent : NSObject
{
    long m_TotalTimeUs;
    int m_TotalTags;
    int m_TotalRounds;
}

- (long)getTotalTimeUs;
- (void)setTotalTimeUs:(long)val;
- (int)getTotalTags;
- (void)setTotalTags:(int)val;
- (int)getTotalRounds;
- (void)setTotalRounds:(int)val;

@end
```

Table 3-44 *srfidOperEndSummaryEvent Variable Descriptions*

Variable Name	Description
m_TotalTimeUs	Total time in Us.
m_TotalTags	Total tags read.
m_TotalRounds	Total rounds of inventory ran.

srfidAttribute

Description

Used to set and get the value and type of a specific attribute of a specific RFID reader.

```
@interface srfidAttribute : NSObject
{
NSMutableString *m_AttrType;
int m_AttrNum;
id m_AttrVal;
int m_Offset;
int m_PropertyVal;
int m_Length;
}
- (int)getAttrNum;
- (void)setAttrNum:(int)val;
- (id)getAttrVal;
- (void)setAttrVal:(id)val;
- (int)getOffset;
- (void)setOffset:(int)val;
- (NSString*)getAttrType;
- (void)setAttrType:(NSString*)val;
- (int)getPropertyVal;
- (void)setPropertyVal:(int)val;
- (int)getLength;
- (void)setLength:(int)val;
@end
```

Table 3-45 *srfidAttribute Variable Descriptions*

Variable Name	Description
m_AttrType	Specifies attribute type.
m_AttrNum	Specifies attribute number.
m_AttrVal	Specifies attribute value.
m_Offset	Specifies attribute offset.
m_PropertyVal	Specifies attribute property value.
m_Length	Specifies attribute length.

Functions

API functions are defined by the srfidISdkApi Objective C protocol. A specific object which implements the protocol is accessible via the createRfidSdkApiInstance method of the srfidSdkFactory class. See *srfidISdkApi on page 3-16* and *srfidSdkFactory on page 3-20* for details.

srfidGetSdkVersion

Description

Get version of the SDK.

```
(NSString*) srfidGetSdkVersion;
```

Parameters

None.

Return Values

SDK version as NSString autoreleased object.

srfidSetDelegate

Description

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

```
(SRFID_RESULT) srfidSetDelegate: (id<srfidISdkApiDelegate>) delegate;
```

Parameters

`(id<srfidISdkApiDelegate>)delegate`

[in] An object which conforms to srfidISdkApiDelegate protocol.

Return Values

SRFID_RESULT_SUCCESS

The receiver of SDK notifications was registered successfully.

srifdSetOperationalMode

Description

Configure operating mode of the SDK.

```
- (SRFID_RESULT) srfidSetOperationalMode: (int)operationalMode;
```

Parameters

`(int)operationalMode`

[in] Identifier of requested operating mode.

(continued on next page)

Return Values

SRFID_RESULT_SUCCESS

The requested operating mode of the SDK was configured successfully.

SRFID_RESULT_FAILURE

Invalid parameters.

✓ ***NOTES***

- If operating mode of the SDK is not configured intentionally, the SDK will remain disabled and will not be able to communicate with RFID readers in either "iOS BT MFi" or "iOS BT LE" mode.
- Selection of new operating mode results in:
  - Disconnection of active incompatible RFID readers (e.g., "iOS BT LE" RFID readers in SRFID_OPMODE_MFI mode), removing available incompatible RFID readers and providing corresponding notifications if these notifications are enabled.
  - Performing discovery of available RFID readers compatible with requested operating mode (if "Available readers detection" option is enabled) and providing corresponding notifications if these notifications are enabled.

srfidSubsribeForEvents

Description

Enables providing of notification of requested types.

```
- (SRFID_RESULT) srfidSubsribeForEvents: (int) sdkEventsMask;
```

Parameters

`(int)sdkEventsMask`

[in] Subscription option flags.

Return Values

SRFID_RESULT_SUCCESS

Subscription was performed successfully.

srfidUnsubsribeForEvents

Description

Disables providing of notification of requested types.

```
- (SRFID_RESULT) srfidUnsubsribeForEvents: (int) sdkEventsMask;
```

Parameters

`(int)sdkEventsMask`

[in] Unsubscription option flags.

Return Values

SRFID_RESULT_SUCCESS

Unsubscription was performed successfully.

srfidGetAvailableReadersList

Description

Request the list of available RFID readers.

```
(SRFID_RESULT) srfidGetAvailableReadersList:
(NSMutableArray**) availableReadersList;
```

Parameters

`(NSMutableArray**)availableReadersList`

[out] Pointer to NSMutableArray object intended for storage of srfidReaderInfo objects that represent available RFID readers.

Return Values

SRFID_RESULT_SUCCESS

Discovery procedure was performed and availableReadersList parameter is filled with available RFID readers.

SRFID_RESULT_INVALID_PARAMS

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

✓ ***NOTES***

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

srfidGetActiveReadersList

Description

Requests the list of active RFID readers.

```
- (SRFID_RESULT) srfidGetActiveReadersList: (NSMutableArray**)activeReadersList;
```

Parameters

`(NSMutableArray**)activeReadersList`

[out] Pointer to NSMutableArray object intended for storage of srfidReaderInfo objects that represent active RFID readers.

Return Values

SRFID_RESULT_SUCCESS

Discovery procedure was performed and activeReadersList parameter is filled with active RFID readers.

SRFID_RESULT_INVALID_PARAMS

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

✓ ***NOTES***

- The caller is responsible for allocation, initialization and deallocation of NSMutableArray object.
- All objects from received NSMutableArray object are removed.
- Received NSMutableArray object is filled with srfidReaderInfo objects that represents active RFID readers.

srfidEstablishCommunicationSession

Description

Request to establish communication session with a specific available RFID reader.

```
- (SRFID_RESULT) srfidEstablishCommunicationSession: (int) readerID;
```

Parameters

`(int)readerID`

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

Return Values

SRFID_RESULT_SUCCESS

The communication session was established successfully.

SRFID_RESULT_FAILURE

The communication session was not established.

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

srfidTerminateCommunicationSession

Description

Requests to terminate communication session with a specific active RFID reader.

```
- (SRFID_RESULT) srfidTerminateCommunicationSession: (int) readerID;
```

Parameters

`(int)readerID`
[in] Unique identifier of a specific RFID reader assigned by SDK.

Return Values

SRFID_RESULT_SUCCESS

The communication session was terminated successfully.

SRFID_RESULT_FAILURE

The communication session was not terminated.

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

srfidEstablishAsciiConnection

Description

Request to establish an ASCII protocol level connection with a particular RFID reader.

```
- (SRFID_RESULT) srfidEstablishAsciiConnection:(int)readerID
aPassword:(NSString*)password;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(NSString*)password`

[in] Password required for establishment of ASCII protocol level connection.

Return Values

SRFID_RESULT_SUCCESS

ASCII protocol level connection has been established successfully.

SRFID_RESULT_FAILURE

SDK has failed to start establish an ASCII protocol level connection.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_WRONG_ASCII_PASSWORD

ASCII protocol level connection has not been established due to incorrect connection password.

✓ ***NOTES***

- ASCII protocol level connection is established through sending connect ASCII protocol command.
- If password parameter is not specified (e.g., is nil value or an empty string) corresponding.password parameter of inventory ASCII protocol command is not be specified.

srfidEnableAvailableReadersDetection

Description

Request to enable/disable "Available readers detection" option.

```
- (SRFID_RESULT) srfidEnableAvailableReadersDetection: (BOOL) enable;
```

Parameters

`(BOOL)enable`

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

YES

Requests to enable "Available readers detection" option.

NO

Requests to disable "Available readers detection" option.

Return Values

SRFID_RESULT_SUCCESS

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

✓ ***NOTES***

- The SDK will perform discovery of available RFID readers once the option is enabled and will provide corresponding notifications if that notifications are enabled.
- If the option is enabled the SDK will detect connection and disconnection of RFID readers operating in "iOS BT MFi" mode in both foreground and background execution modes of a specific application linked with SDK with providing of corresponding notifications if that notifications are enabled.
- If the option is enabled the SDK will detect appearance and disappearance of RFID readers operating in "iOS BT LE" mode in only foreground execution mode of a specific application linked with SDK through periodic discovery operation with providing of corresponding notifications if that notifications are enabled.

srfidEnableAutomaticSessionReestablishment

Description

Requests to enable/disable "Automatic communication session reestablishment" option.

```
- (SRFID_RESULT) srfidEnableAutomaticSessionReestablishment: (BOOL) enable;
```

Parameters

`(BOOL)enable`

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

YES

Requests to enable "Automatic communication session reestablishment" option.

NO

Requests to disable "Automatic communication session reestablishment" option.

Return Values

SRFID_RESULT_SUCCESS

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

✓ ***NOTES***

- If the option is enabled the SDK will automatically establish communication session with the last active RFID reader that had unexpectedly disappeared once the RFID reader will be recognized as available:
  - The RFID reader could be recognized as available automatically by SDK if "Available readers detection" option is enabled.
  - The RFID reader could be recognized as available during discovery procedure requested by srfidGetAvailableReadersList API.
- "Session Established" notification will be provided once the communication session is established, if this type of notification is enabled.

srfidStartRapidRead

Description

Request to start rapid read operation on a specific RFID reader.

```
- (SRFID_RESULT) srfidStartRapidRead:(int)readerID
aReportConfig:(srfidReportConfig*)reportConfig
aAccessConfig:(srfidAccessConfig*)accessConfig
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

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

`srfidReportConfig*)reportConfig`

[in] Report parameters for rapid read operation.

`(srfidAccessConfig*)accessConfig`

[in] Access parameters for rapid read operation.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Rapid read operation was started successfully.

SRFID_RESULT_FAILURE

SDK failed to start rapid read operation.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by the readerID parameter was not active or available.

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

NOTES

- Rapid read operation is started through sending inventory ASCII protocol command.
- Rapid read operation is started through sending inventory ASCII protocol command.
- If reportConfig and/or accessConfig parameters are not specified (e.g., are nil values) corresponding parameters of inventory ASCII protocol command will not be specified.
- "Read Event" notification will be provided for each RFID tag.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidStopRapidRead

Description

Request to stop rapid read operation on a particular RFID reader.

```
- (SRFID_RESULT) srfidStopRapidRead:(int)readerID
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Rapid read operation has been stopped successfully.

SRFID_RESULT_FAILURE

SDK has failed to stop rapid read operation.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ ***NOTES***

- Rapid read operation is stopped through sending abort ASCII protocol command.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidStartInventory

Description

Requests to start inventory operation on a particular RFID reader.

```
- (SRFID_RESULT) srfidStartInventory:(int)readerID
aMemoryBank:(SRFID_MEMORYBANK)memoryBankId
aReportConfig:(srfidReportConfig*)reportConfig
aAccessConfig:(srfidAccessConfig*)accessConfig
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(SRFID_MEMORYBANK)memoryBankId`

[in] Identifier of memory bank to be used.

`(srfidReportConfig*)reportConfig`

[in] Report parameters for inventory operation.

`(srfidAccessConfig*)accessConfig`

[in] Access parameters for inventory operation.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Inventory operation has been started successfully.

SRFID_RESULT_FAILURE

SDK has failed to start inventory operation.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

Invalid parameters (e.g., an identifier of memory bank is not specified).

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ ***NOTES***

- Inventory operation is started through sending read ASCII protocol command.
- If reportConfig and/or accessConfig parameters are not specified (e.g., are nil values) corresponding parameters of read ASCII protocol command will not be specified.
- "Read event" notification will be provided for each RFID tag.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidStopInventory

Description

Request to stop inventory operation on a particular RFID reader.

```
- (SRFID_RESULT) srfidStopInventory:(int)readerID
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Inventory operation has been stopped successfully.

SRFID_RESULT_FAILURE

SDK has failed to stop inventory operation.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ ***NOTES***

- Inventory operation is stopped through sending abort ASCII protocol command.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidGetSupportedLinkProfiles

Description

Request to receive information regarding all supported link profiles (RF modes) from a particular RFID reader.

```
- (SRFID_RESULT) srfidGetSupportedLinkProfiles:(int)readerID
aLinkProfilesList:(NSMutableArray**)linkProfilesList
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(NSMutableArray**)linkProfilesList`

[out] Pointer to NSMutableArray object intended for storage of srfidLinkProfile objects that represent supported link profiles.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Information about supported link profiles has been received successfully.

SRFID_RESULT_FAILURE

SDK has failed to receive the information about supported link profiles.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

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

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ ***NOTES***

- The caller is responsible for allocation, initialization and deallocation of NSMutableArray object.
- All objects from received NSMutableArray object are removed.
- Information is requested through sending getsupportedlinkprofiles ASCII protocol command.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidGetAntennaConfiguration

Description

Request to receive configured antenna parameters from a particular RFID reader.

```
- (SRFID_RESULT) srfidGetAntennaConfiguration:(int)readerID
aAntennaConfiguration:(srfidAntennaConfiguration**)antennaConfiguration
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(srfidAntennaConfiguration**)antennaConfiguration`

[out] Pointer to srfidAntennaConfiguration object intended for storage of received antenna parameters.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Antenna parameters have been received successfully.

SRFID_RESULT_FAILURE

SDK has failed to receive antenna parameters.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

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

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

NOTES

- The caller is responsible for allocation, initialization and deallocation of srfidAntennaConfiguration object.
- Information is requested through sending setantennaconfiguration ASCII protocol command with noexec parameter.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidSetAntennaConfiguration

Description

Request to set particular antenna parameters on a particular RFID reader.

```
- (SRFID_RESULT) srfidSetAntennaConfiguration:(int)readerID
aAntennaConfiguration:(srfidAntennaConfiguration*)antennaConfiguration
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(srfidAntennaConfiguration*)antennaConfiguration`

[in] srfidAntennaConfiguration object which contains antenna parameters to be applied.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Antenna parameters have been applied successfully.

SRFID_RESULT_FAILURE

SDK failed to apply antenna parameters.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

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

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ ***NOTES***

- Applied parameters are lost after next power down. srfidSaveReaderConfiguration API shall be used to make the applied configuration persistent.
- Antenna parameters are applied through sending setantennaconfiguration ASCII protocol command.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidGetDpoConfiguration

Description

Request to receive configured dynamic power optimization parameters from a particular RFID reader.

```
- (SRFID_RESULT) srfidGetDpoConfiguration:(int)readerID aDpoConfiguration:(
srfidDynamicPowerConfig**)dpoConfiguration aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(srfidDynamicPowerConfig **) dpoConfiguration`

[out] Pointer to srfidDynamicPowerConfig object intended for storage of received dynamic power optimization parameters.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Dynamic power optimization parameters have been received successfully.

SRFID_RESULT_FAILURE

SDK has failed to receive dynamic optimization parameters.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

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

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ ***NOTES***

- The caller is responsible for allocation, initialization and deallocation of srfidDynamicPowerConfig object.
- Information is requested through sending setdynamicpower ASCII protocol command with noexec parameter.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidSetDpoConfiguration

Description

Request to set particular dynamic power optimization parameters on a particular RFID reader.

```
- (SRFID_RESULT) srfidSetDpoConfiguration:(int)readerID aDpoConfiguration:(
srfidDynamicPowerConfig *)dpoConfiguration aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(srfidDynamicPowerConfig*)dpoConfiguration`

[in] srfidSetDpoConfiguration object which contains dynamic power optimization parameters to be applied.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Dynamic power optimization parameters have been applied successfully.

SRFID_RESULT_FAILURE

SDK failed to apply dynamic power optimization parameters.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

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

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ ***NOTES***

- Applied parameters are lost after next power down. srfidSaveReaderConfiguration API shall be used to make the applied configuration persistent.
- Dynamic power optimization parameters are applied through sending setdynamicpower ASCII protocol command.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidGetSingulationConfiguration

Description

Request to receive configured singulation parameters from a particular RFID reader.

```
- (SRFID_RESULT) srfidGetSingulationConfiguration:(int)readerID
aSingulationConfig:(srfidSingulationConfig**)singulationConfig
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(srfidSingulationConfig**)singulationConfig`

[out] Pointer to srfidSingulationConfig object which contains received singulation parameters.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Singulation parameters have been received successfully.

SRFID_RESULT_FAILURE

SDK has failed to receive singulation parameters.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

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

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ ***NOTES***

- The caller is responsible for allocation, initialization and deallocation of srfidSingulationConfig object.
- Information is requested through sending setqueryparams ASCII protocol command with noexec parameter.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidSetSingulationConfiguration

Description

Request to set particular singulation parameters on a particular RFID reader.

```
- (SRFID_RESULT) srfidSetSingulationConfiguration:(int)readerID
aSingulationConfig:(srfidSingulationConfig*)singulationConfig
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(srfidSingulationConfig*)singulationConfig`

[in] srfidSingulationConfig object which contains singulation parameters to be applied.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Singulation parameters have been applied successfully.

SRFID_RESULT_FAILURE

SDK has failed to apply singulation parameters.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

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

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ ***NOTES***

- Applied parameters are lost after next power down. srfidSaveReaderConfiguration API shall be used to make the applied configuration persistent.
- Singulation parameters are applied through sending setqueryparams ASCII protocol command.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidGetTagReportConfiguration

Description

Request to receive configured report parameters from a particular RFID reader.

```
- (SRFID_RESULT) srfidGetTagReportConfiguration:(int)readerID
aTagReportConfig:(srfidTagReportConfig**)reportConfig
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(srfidTagReportConfig**)reportConfig`

[out] Pointer to srfidTagReportConfig object intended for storage of received report parameters.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Report parameters have been received successfully.

SRFID_RESULT_FAILURE

SDK has failed to receive report configuration.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

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

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ ***NOTES***

- The caller is responsible for allocation, initialization and deallocation of srfidTagReportConfig object.
- Information is requested through sending setreportconfig ASCII protocol command with noexec parameter.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidSetTagReportConfiguration

Description

Requests to set particular report parameters on a particular RFID reader.

```
- (SRFID_RESULT) srfidSetTagReportConfiguration:(int)readerID
aTagReportConfig:(srfidTagReportConfig*)reportConfig
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(srfidTagReportConfig*)reportConfig`

[in] srfidTagReportConfig object which contains report parameters to be applied.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface

Return Values

SRFID_RESULT_SUCCESS

Report parameters have been applied successfully.

SRFID_RESULT_FAILURE

SDK has failed to apply report parameters.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

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

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ ***NOTES***

- Applied parameters are lost after next power down. srfidSaveReaderConfiguration API shall be used to make the applied configuration persistent.
- Report parameters are applied through sending setreportconfig ASCII protocol command.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidSaveReaderConfiguration

Description

Request to store current configuration of a particular RFID reader.

```
- (SRFID_RESULT) srfidSaveReaderConfiguration:(int)readerID
aSaveCustomDefaults:(BOOL)saveCustomDefaults aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(BOOL)saveCustomDefaults`

[in] Indicates whether current configuration shall be stored to flash or to custom defaults area.

YES

Current configuration shall be stored to custom defaults area.

NO

Current configuration shall be stored to flash to be persistent over power down and power up cycles.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Current configuration has been stored successfully.

SRFID_RESULT_FAILURE

SDK has failed to store current configuration.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ ***NOTES***

- Current configuration is stored through sending changeconfig ASCII protocol command with saveconfig parameter or savecustomdefaults parameter.
- The configuration stored to custom defaults can be restored with srfidRestoreReaderConfiguration API.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidRestoreReaderConfiguration

Description

Request to restore factory default or custom default configuration of a particular RFID reader.

```
- (SRFID_RESULT) srfidRestoreReaderConfiguration:(int)readerID
aRestoreFactoryDefaults:(BOOL)restoreFactoryDefaults
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(BOOL)restoreFactoryDefaults`

[in] Indicates whether configuration shall be restored from custom defaults area or from factory defined values.

YES

Configuration shall be restored from factory defined values.

NO

Configuration shall be restored from custom defaults area.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Configuration has been restored successfully.

SRFID_RESULT_FAILURE

SDK has failed to restore configuration.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

NOTES

- Configuration is restored through sending changeconfig ASCII protocol command with restorefactorydefaults parameter or restorecustomdefaults parameter.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidGetReaderVersionInfo

Description

Request to receive software version information from a particular RFID reader.

```
- (SRFID_RESULT) srfidGetReaderVersionInfo:(int)readerID
aReaderVersionInfo:(srfidReaderVersionInfo**)readerVersionInfo
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(srfidReaderVersionInfo**)readerVersionInfo`

[out] Pointer to srfidReaderVersionInfo object intended for storage of received version information.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Version information has been received successfully.

SRFID_RESULT_FAILURE

SDK has failed to receive version information.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

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

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ ***NOTES***

- The caller is responsible for allocation, initialization and deallocation of srfidReaderVersionInfo object.
- Information is requested through sending getversion ASCII protocol command.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidGetReaderCapabilitiesInfo

Description

Request to receive capabilities related information from a particular RFID reader.

```
- (SRFID_RESULT) srfidGetReaderCapabilitiesInfo:(int)readerID
aReaderCapabilitiesInfo:(srfidReaderCapabilitiesInfo**)readerCapabilitiesInfo
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(srfidReaderCapabilitiesInfo**)readerCapabilitiesInfo`

[out] Pointer to srfidReaderCapabilitiesInfo object intended for storage of received capabilities related information.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Capabilities information has been received successfully.

SRFID_RESULT_FAILURE

SDK has failed to receive capabilities information.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

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

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

NOTES

- The caller is responsible for allocation, initialization and deallocation of srfidReaderCapabilitiesInfo object.
- Information is requested through sending getcapabilities ASCII protocol command.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidGetStartTriggerConfiguration

Description

Request to receive start trigger parameters from a particular RFID reader.

```
- (SRFID_RESULT) srfidGetStartTriggerConfiguration:(int)readerID
aStartTriggeConfig:(srfidStartTriggerConfig**)triggerConfig
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(srfidStartTriggerConfig**)triggerConfig`

[out] Pointer to srfidStartTriggerConfig object intended for storage of received start trigger parameters.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Start trigger parameters have been received successfully.

SRFID_RESULT_FAILURE

SDK has failed to receive start trigger parameters.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

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

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ ***NOTES***

- The caller is responsible for allocation, initialization and deallocation of srfidStartTriggerConfig object.
- Information is requested through sending setstarttrigger ASCII protocol command with noexec parameter.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidSetStartTriggerConfiguration

Description

Request to set start trigger parameters on a particular RFID reader.

```
- (SRFID_RESULT) srfidSetStartTriggerConfiguration:(int)readerID
aStartTriggeConfig:(srfidStartTriggerConfig*)triggerConfig
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(srfidStartTriggerConfig*)triggerConfig`

[in] Pointer to srfidStartTriggerConfig object which contains start trigger parameters to be applied.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Start trigger parameters have been applied successfully.

SRFID_RESULT_FAILURE

SDK has failed to apply start trigger parameters.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

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

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ ***NOTES***

- Applied parameters are lost after next power down. srfidSaveReaderConfiguration API shall be used to make the applied configuration persistent.
- Start trigger parameters are applied through sending setstarttrigger ASCII protocol command.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidGetStopTriggerConfiguration

Description

Request to receive stop trigger parameters from a particular RFID reader.

```
- (SRFID_RESULT) srfidGetStopTriggerConfiguration:(int)readerID
aStopTriggeConfig:(srfidStopTriggerConfig**)triggerConfig
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(srfidStopTriggerConfig**)triggerConfig`

[out] Pointer to srfidStopTriggerConfig object intended for storage of received stop trigger parameters.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Stop trigger parameters have been received successfully.

SRFID_RESULT_FAILURE

SDK has failed to receive stop trigger parameters.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

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

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ ***NOTES***

- The caller is responsible for allocation, initialization and deallocation of srfidStopTriggerConfig object.
- Information is requested through sending setstoptrigger ASCII protocol command with noexec parameter.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidSetStopTriggerConfiguration

Description

Request to set stop trigger parameters on a particular RFID reader.

```
- (SRFID_RESULT) srfidSetStopTriggerConfiguration:(int)readerID
aStopTriggeConfig:(srfidStopTriggerConfig*)triggerConfig
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(srfidStopTriggerConfig*)triggerConfig`

[in] Pointer to srfidStopTriggerConfig object which contains stop trigger parameters to be applied.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface

Return Values

SRFID_RESULT_SUCCESS

Stop trigger parameters have been applied successfully.

SRFID_RESULT_FAILURE

SDK has failed to apply stop trigger parameters.

SRFID_READER_NOT_AVAILABLE

The operation was not performed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

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

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ ***NOTES***

- Applied parameters are lost after next power down. srfidSaveReaderConfiguration API shall be used to make the applied configuration persistent.
- Stop trigger parameters are applied through sending setstoptrigger ASCII protocol command.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidGetSupportedRegions

Description

Request to receive information regarding all supported regions from a particular RFID reader.

```
- (SRFID_RESULT) srfidGetSupportedRegions:(int)readerID
aSupportedRegions:(NSMutableArray**)supportedRegionsList
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(NSMutableArray**)supportedRegionsList`

[out] Pointer to NSMutableArray object intended for storage of srfidRegionInfo objects that represent supported regions.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Information about supported regions has been received successfully.

SRFID_RESULT_FAILURE

SDK has failed to receive the information about supported regions.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

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

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

NOTES

- The caller is responsible for allocation, initialization and deallocation of NSMutableArray object.
- All objects from received NSMutableArray object are removed.
- Information is requested through sending getallsupportedregions ASCII protocol command.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidGetRegionInfo

Description

Request to receive detailed information regarding one of supported regions from a particular RFID reader.

```
- (SRFID_RESULT) srfidGetRegionInfo:(int)readerID aRegionCode:(NSString*)regionCode
aSupportedChannels:(NSMutableArray**)supportedChannelsList
aHoppingConfigurable:(BOOL*)hoppingConfigurable
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(NSString*)regionCode`

[in] Unique code of a particular region.

`(NSMutableArray**)supportedChannelsList`

[out] Pointer to NSMutableArray object intended for storage of NSString objects that represent supported channels for the region specified by regionCode parameter.

`(BOOL*)hoppingConfigurable`

[out] Pointer to BOOL variable intended for storage hopping related configuration (enabled/disabled) for the region specified by regionCode parameter.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Information about a particular region has been received successfully.

SRFID_RESULT_FAILURE

SDK has failed to receive the information about a particular region.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

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

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ ***NOTES***

- The caller is responsible for allocation, initialization and deallocation of NSMutableArray object.
- All objects from received NSMutableArray object are removed.
- Information is requested through sending getregion ASCII protocol command.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidGetRegulatoryConfig

Description

Request to receive regulatory parameters from a particular RFID reader.

```
- (SRFID_RESULT) srfidGetRegulatoryConfig:(int)readerID
aRegulatoryConfig:(srfidRegulatoryConfig**)regulatoryConfig
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(srfidRegulatoryConfig**)regulatoryConfig`

[out] Pointer to srfidRegulatoryConfig object intended for storage of received regulatory parameters.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Regulatory parameters have been received successfully.

SRFID_RESULT_FAILURE

SDK has failed to receive regulatory parameters.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

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

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ ***NOTES***

- The caller is responsible for allocation, initialization and deallocation of srfidRegulatoryConfig object.
- Information is requested through sending setregulatory ASCII protocol command with noexec parameter.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidSetRegulatoryConfig

Description

Request to set regulatory parameters on a particular RFID reader.

```
- (SRFID_RESULT) srfidSetRegulatoryConfig:(int)readerID
aRegulatoryConfig:(srfidRegulatoryConfig*)regulatoryConfig
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(srfidRegulatoryConfig*)regulatoryConfig`

[in] Pointer to srfidRegulatoryConfig object which contains regulatory parameters to be applied.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Regulatory parameters have been applied successfully.

SRFID_RESULT_FAILURE

SDK has failed to apply regulatory parameters.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

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

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ ***NOTES***

- The applied configuration is lost after next power down. srfidSaveReaderConfiguration API shall be used to make the applied configuration persistent.
- Regulatory parameters are applied through sending setregulatory ASCII protocol command.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidGetBeeperConfig

Description

Request to receive beeper configuration from a particular RFID reader.

```
- (SRFID_RESULT) srfidGetBeeperConfig:(int)readerID
aBeeperConfig:(SRFID_BEEPERCONFIG*)beeperConfig
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(SRFID_BEEPERCONFIG*)beeperConfig`

[out] Pointer to SRFID_BEEPERCONFIG variable intended for storage of received beeper configuration.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Regulatory parameters have been received successfully.

SRFID_RESULT_FAILURE

SDK has failed to receive regulatory parameters.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

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

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ ***NOTES***

- Information is requested through sending getattrinfo ASCII protocol command.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidSetBeeperConfig

Description

Request to set beeper configuration on a particular RFID reader.

```
- (SRFID_RESULT) srfidSetBeeperConfig:(int)readerID
aBeeperConfig:(SRFID_BEEPERCONFIG)beeperConfig
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(SRFID_BEEPERCONFIG*)beeperConfig`

[in] Required beeper configuration.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Regulatory parameters have been applied successfully.

SRFID_RESULT_FAILURE

SDK has failed to apply regulatory parameters.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

Invalid parameter.

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

NOTES

- The applied configuration is lost after next power down. srfidSaveReaderConfiguration API shall be used to make the applied configuration persistent.
- Beeper configuration is applied through sending setattr ASCII protocol command.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidGetPreFilters

Description

Request to receive information regarding configured prefilters (select records) from a particular RFID reader.

```
- (SRFID_RESULT) srfidGetPreFilters:(int)readerID
aPreFilters:(NSMutableArray**)filtersList aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(NSMutableArray**)filtersList`

[out] Pointer to NSMutableArray object intended for storage of srfidPreFilter objects that represent configured prefilters.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Information about configured prefilters has been received successfully.

SRFID_RESULT_FAILURE

SDK has failed to receive information about configured prefilters.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

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

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

NOTES

- The caller is responsible for allocation, initialization and deallocation of NSMutableArray object.
- All objects from received NSMutableArray object are removed.
- Information is requested through sending setselectrecords ASCII protocol command with noexec parameter.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidSetPreFilters

Description

Request to configure a particular set of prefilters (select records) on a particular RFID reader.

```
- (SRFID_RESULT) srfidSetPreFilters:(int)readerID
aPreFilters:(NSMutableArray*)filtersList aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(NSMutableArray*)filtersList`

[in] Pointer to NSMutableArray object filled with srfidPreFilter objects that represent prefilters to be configured.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Prefilter has been configured successfully.

SRFID_RESULT_FAILURE

SDK has failed to configure prefilter.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

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

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

NOTES

- Prefilters are configured through sending setselectrecords ASCII protocol command.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidStartTagLocationing

Description

Request to start tag locationing operation on a particular RFID reader.

```
- (SRFID_RESULT) srfidStartTagLocationing:(int)readerID aTagEpcId:(NSString*)epcID
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(NSString*)epcID`

[in] Pointer to NSString object which specifies EPC ID of the tag to be located.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Operation has been started successfully.

SRFID_RESULT_FAILURE

SDK has failed to start operation.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

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

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ ***NOTES***

- Tag locationing operation is started through sending locatetag ASCII protocol command.
- "Proximity Event" notifications will be provided during on-going tag locationing operation.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidStopTagLocationing

Description

Request to stop tag locationing operation on a particular RFID reader.

```
- (SRFID_RESULT) srfidStopTagLocationing:(int)readerID
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Operation has been stopped successfully.

SRFID_RESULT_FAILURE

SDK has failed to stop operation.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

NOTES

- Tag locationing operation is stopped through sending abort ASCII protocol command.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidReadTag

Description

Request to perform read operation for a particular tag with a particular RFID reader.

```
- (SRFID_RESULT) srfidReadTag:(int)readerID aTagID:(NSString*)tagID
aAccessTagData:(srfidTagData**)accessTagData
aMemoryBank:(SRFID_MEMORYBANK)memoryBankID aOffset:(short)offset aLength:(short)length
aPassword:(long)password aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(NSString*)tagID`

[in] Identifier of the tag on that the read operation shall be performed.

`(srfidTagData**)accessTagData`

[out] Pointer to the srfidTagData object intended for storing results of the performed read operation.

`(SRFID_MEMORYBANK)memoryBankID`

[in] Identifier of a particular memory bank on that read operation shall be performed.

`(short)offset`

[in] Number of words offsetted from beginning of memory bank from where the read operation shall be performed.

`(short)length`

[in] Number of words to read.

`(long)password`

[in] Access password.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Operation has been performed successfully.

SRFID_RESULT_FAILURE

SDK has failed to perform operation.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

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

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

(continued on next page)

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ ***NOTES***

- The caller is responsible for allocation, initialization and deallocation of srfidTagData object.
- The operation is performed though a following set of ASCII protocol commands:
- Current start trigger configuration is requested though sending setstarttrigger ASCII protocol command with noexec parameter and is stored by the SDK.
- Current stop trigger configuration is requested though sending setstoptrigger ASCII protocol command with noexec parameter and is stored by the SDK.
- Criteria for access operation is configured in accordance with tagID parameter though sending setaccessfilter ASCII protocol command.
- Start trigger configuration is updated to ignore hand-held trigger, enable operation start without any delay and disable repeat monitoring though sending setstarttrigger ASCII protocol command.
- Stop trigger configuration is updated to ignore hand-held trigger, enable operation stop on single completed access round or on 5000 milliseconds timeout though sending setstoptrigger ASCII protocol command.
- Start of access operation is requested though sending read ASCII protocol command.
- After completion of access operation previous start and stop trigger configurations are restored though sending setstarttrigger and setstoptrigger ASCII protocol commands.
- "Status Event" and "Read Event" notifications are disabled during the on-going access operation.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidReadTag (with Access Criteria)

Description

Request to perform read operation for a particular tag with a particular RFID reader.

```
- (SRFID_RESULT) srfidReadTag:(int)readerID
aAccessCriteria:(srfidAccessCriteria*)accessCriteria
aAccessTagData:(srfidTagData**)accessTagData
aMemoryBank:(SRFID_MEMORYBANK)memoryBankID aOffset:(short)offset aLength:(short)length
aPassword:(long)password aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(srfidAccessCriteria*)accessCriteria`

[in] Access criteria to identify the Tag on which the read operation needs to be carried out by the SDK. Using the Access Criteria a tag can be chosen with one of the memory bank data.

`(srfidTagData**)accessTagData`

[out] Pointer to the srfidTagData object intended for storing results of the performed read operation.

`(SRFID_MEMORYBANK)memoryBankID`

[in] Identifier of a particular memory bank on that read operation shall be performed.

`(short)offset`

[in] Number of words offsetted from beginning of memory bank from where the read operation shall be performed.

`(short)length`

[in] Number of words to read.

`(long)password`

[in] Access password.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Operation has been performed successfully.

SRFID_RESULT_FAILURE

SDK has failed to perform operation.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

Invalid parameter (e.g. nil pointer).

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

(continued on next page)

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

NOTES

- The caller is responsible for allocation, initialization and deallocation of srfidTagData object.
- The operation is performed though a following set of ASCII protocol commands:
- Current start trigger configuration is requested though sending setstarttrigger ASCII protocol command with noexec parameter and is stored by the SDK.
- Current stop trigger configuration is requested though sending setstoptrigger ASCII protocol command with noexec parameter and is stored by the SDK.
- Criteria for access operation is configured in accordance with the accessCriteria parameter though sending setaccessfilter ASCII protocol command.
- Start trigger configuration is updated to ignore hand-held trigger, enable operation start without any delay and disable repeat monitoring though sending setstarttrigger ASCII protocol command.
- Stop trigger configuration is updated to ignore hand-held trigger, enable operation stop on single completed access round or on 5000 milliseconds timeout though sending setstoptrigger ASCII protocol command.
- Start of access operation is requested though sending read ASCII protocol command.
- After completion of access operation previous start and stop trigger configurations are restored though sending setstarttrigger and setstoptrigger ASCII protocol commands.
- "Status Event" and "Read Event" notifications are disabled during the on-going access operation.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidWriteTag

Description

Request to perform write operation for a particular tag with a particular RFID reader.

```
- (SRFID_RESULT) srfidWriteTag:(int)readerID aTagID:(NSString*)tagID
aAccessTagData:(srfidTagData**)accessTagData
aMemoryBank:(SRFID_MEMORYBANK)memoryBankID aOffset:(short)offset aData:(NSString*)data
aPassword:(long)password aDoBlockWrite:(BOOL)blockWrite
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(NSString*)tagID`

[in] Identifier of the tag on that the write operation shall be performed.

`(srfidTagData**)accessTagData`

[out] Pointer to the srfidTagData object intended for storing results of the performed write operation.

`(SRFID_MEMORYBANK)memoryBankID`

[in] Identifier of a particular memory bank on that write operation shall be performed.

`(short)offset`

[in] Number of words offsetted from beginning of memory bank from where the write operation shall be performed.

`(NSString*)data`

[in] Data to be written stored as ASCII HEX string.

`(long)password`

[in] Access password.

`(BOOL)blockWrite`

[in] Specifies whether a block write operation shall be performed.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Operation has been performed successfully.

SRFID_RESULT_FAILURE

SDK has failed to perform operation.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

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

(continued on next page)

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ ***NOTES***

- The caller is responsible for allocation, initialization and deallocation of srfidTagData object.
- The operation is performed though a following set of ASCII protocol commands:
- Current start trigger configuration is requested though sending setstarttrigger ASCII protocol command with noexec parameter and is stored by the SDK.
- Current stop trigger configuration is requested though sending setstoptrigger ASCII protocol command with noexec parameter and is stored by the SDK.
- Criteria for access operation is configured in accordance with tagID parameter though sending setaccessfilter ASCII protocol command.
- Start trigger configuration is updated to ignore hand-held trigger, enable operation start without any delay and disable repeat monitoring though sending setstarttrigger ASCII protocol command.
- Stop trigger configuration is updated to ignore hand-held trigger, enable operation stop on single completed access round or on 5000 milliseconds timeout though sending setstoptrigger ASCII protocol command.
- Start of access operation is requested though sending write ASCII protocol command.
- After completion of access operation previous start and stop trigger configurations are restored though sending setstarttrigger and setstoptrigger ASCII protocol commands.
- "Status Event" and "Read Event" notifications are disabled during the on-going access operation.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidWriteTag (with Access Criteria)

Description

Request to perform write operation for a particular tag with a particular RFID reader.

```
- (SRFID_RESULT) srfidWriteTag:(int)readerID
aAccessCriteria:(srfidAccessCriteria*)accessCriteria
aAccessTagData:(srfidTagData**)accessTagData
aMemoryBank:(SRFID_MEMORYBANK)memoryBankID aOffset:(short)offset aData:(NSString*)data
aPassword:(long)password aDoBlockWrite:(BOOL)blockWrite
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(srfidAccessCriteria*)accessCriteria`

[in] Access criteria to identify the Tag on which the write operation needs to be carried out by the SDK. Using the Access Criteria a tag can be chosen with one of the memory bank data.

`(srfidTagData**)accessTagData`

[out] Pointer to the srfidTagData object intended for storing results of the performed write operation.

`(SRFID_MEMORYBANK)memoryBankID`

[in] Identifier of a particular memory bank on that write operation shall be performed.

`(short)offset`

[in] Number of words offsetted from beginning of memory bank from where the write operation shall be performed.

`(NSString*)data`

[in] Data to be written stored as ASCII HEX string.

`(long)password`

[in] Access password.

`(BOOL)blockWrite`

[in] Specifies whether a block write operation shall be performed.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Operation has been performed successfully.

SRFID_RESULT_FAILURE

SDK has failed to perform operation.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

Invalid parameter (e.g. nil pointer).

(continued on next page)

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ ***NOTES***

- The caller is responsible for allocation, initialization and deallocation of srfidTagData object.
- The operation is performed though a following set of ASCII protocol commands:
- Current start trigger configuration is requested though sending setstarttrigger ASCII protocol command with noexec parameter and is stored by the SDK.
- Current stop trigger configuration is requested though sending setstoptrigger ASCII protocol command with noexec parameter and is stored by the SDK.
- Criteria for access operation is configured in accordance with accessCriteria parameter though sending setaccessfilter ASCII protocol command.
- Start trigger configuration is updated to ignore hand-held trigger, enable operation start without any delay and disable repeat monitoring though sending setstarttrigger ASCII protocol command.
- Stop trigger configuration is updated to ignore hand-held trigger, enable operation stop on single completed access round or on 5000 milliseconds timeout though sending setstoptrigger ASCII protocol command.
- Start of access operation is requested though sending write ASCII protocol command.
- After completion of access operation previous start and stop trigger configurations are restored though sending setstarttrigger and setstoptrigger ASCII protocol commands.
- "Status Event" and "Read Event" notifications are disabled during the on-going access operation.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidKillTag

Description

Request to perform kill operation for a particular tag with a particular RFID reader.

```
- (SRFID_RESULT) srfidKillTag:(int)readerID aTagID:(NSString*)tagID
aAccessTagData:(srfidTagData**)accessTagData aPassword:(long)password
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(NSString*)tagID`

[in] Identifier of the tag on that the kill operation shall be performed.

`(srfidTagData**)accessTagData`

[out] Pointer to the srfidTagData object intended for storing results of the performed kill operation.

`(long)password`

[in] Access password.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Operation has been performed successfully.

SRFID_RESULT_FAILURE

SDK has failed to perform operation.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

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

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

(continued on next page)

✓ ***NOTES***

- The caller is responsible for allocation, initialization and deallocation of srfidTagData object.
- The operation is performed though a following set of ASCII protocol commands:
- Current start trigger configuration is requested though sending setstarttrigger ASCII protocol command with noexec parameter and is stored by the SDK.
- Current stop trigger configuration is requested though sending setstoptrigger ASCII protocol command with noexec parameter and is stored by the SDK.
- Criteria for access operation is configured in accordance with tagID parameter though sending setaccessfilter ASCII protocol command.
- Start trigger configuration is updated to ignore hand-held trigger, enable operation start without any delay and disable repeat monitoring though sending setstarttrigger ASCII protocol command.
- Stop trigger configuration is updated to ignore hand-held trigger, enable operation stop on single completed access round or on 5000 milliseconds timeout though sending setstoptrigger ASCII protocol command.
- Start of access operation is requested though sending kill ASCII protocol command.
- After completion of access operation previous start and stop trigger configurations are restored though sending setstarttrigger and setstoptrigger ASCII protocol commands.
- "Status Event" and "Read Event" notifications are disabled during the on-going access operation.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidKillTag (With Access Criteria)

Description

Request to perform kill operation for a particular tag with a particular RFID reader.

```
- (SRFID_RESULT) srfidKillTag:(int)readerID
aAccessCriteria:(srfidAccessCriteria*)accessCriteria
aAccessTagData:(srfidTagData**)accessTagData aPassword:(long)password
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(srfidAccessCriteria*)accessCriteria`

[in] Access criteria to identify the Tag on which the kill operation needs to be carried out by the SDK. Using the Access Criteria a tag can be chosen with one of the memory bank data.

`(srfidTagData**)accessTagData`

[out] Pointer to the srfidTagData object intended for storing results of the performed kill operation.

`(long)password`

[in] Access password.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Operation has been performed successfully.

SRFID_RESULT_FAILURE

SDK has failed to perform operation.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

Invalid parameter (e.g. nil pointer).

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

(continued on next page)

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ ***NOTES***

- The caller is responsible for allocation, initialization and deallocation of srfidTagData object.
- The operation is performed though a following set of ASCII protocol commands:
- Current start trigger configuration is requested though sending setstarttrigger ASCII protocol command with noexec parameter and is stored by the SDK.
- Current stop trigger configuration is requested though sending setstoptrigger ASCII protocol command with noexec parameter and is stored by the SDK.
- Criteria for access operation is configured in accordance with accessCriteria parameter though sending setaccessfilter ASCII protocol command.
- Start trigger configuration is updated to ignore hand-held trigger, enable operation start without any delay and disable repeat monitoring though sending setstarttrigger ASCII protocol command.
- Stop trigger configuration is updated to ignore hand-held trigger, enable operation stop on single completed access round or on 5000 milliseconds timeout though sending setstoptrigger ASCII protocol command.
- Start of access operation is requested though sending kill ASCII protocol command.
- After completion of access operation previous start and stop trigger configurations are restored though sending setstarttrigger and setstoptrigger ASCII protocol commands.
- "Status Event" and "Read Event" notifications are disabled during the on-going access operation.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidLockTag

Description

Request to perform lock operation for a particular tag with a particular RFID reader.

```
- (SRFID_RESULT) srfidLockTag:(int)readerID aTagID:(NSString*)tagID
aAccessTagData:(srfidTagData**)accessTagData
aMemoryBank:(SRFID_MEMORYBANK)memoryBankID
aAccessPermissions:(SRFID_ACCESSPERMISSION)accessPermissions aPassword:(long)password
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(NSString*)tagID`

[in] Identifier of the tag on that the write operation shall be performed.

`(srfidTagData**)accessTagData`

[out] Pointer to the srfidTagData object intended for storing results of the performed lock operation.

`(SRFID_MEMORYBANK)memoryBankID`

[in] Identifier of a particular memory bank on that lock operation shall be performed.

`(SRFID_ACCESSPERMISSION)accessPermissions`

[in] Lock action parameter.

`(long)password`

[in] Access password.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Operation has been performed successfully.

SRFID_RESULT_FAILURE

SDK has failed to perform operation.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

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

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

(continued on next page)

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ ***NOTES***

- The caller is responsible for allocation, initialization and deallocation of srfidTagData object.
- Lock action parameter is applied to the specified by memoryBankID parameter memory bank or to kill and access password fields if memory bank is not specified.
- The operation is performed though a following set of ASCII protocol commands:
- Current start trigger configuration is requested though sending setstarttrigger ASCII protocol command with noexec parameter and is stored by the SDK.
- Current stop trigger configuration is requested though sending setstoptrigger ASCII protocol command with noexec parameter and is stored by the SDK.
- Criteria for access operation is configured in accordance with tagID parameter though sending setaccessfilter ASCII protocol command.
- Start trigger configuration is updated to ignore hand-held trigger, enable operation start without any delay and disable repeat monitoring though sending setstarttrigger ASCII protocol command.
- Stop trigger configuration is updated to ignore hand-held trigger, enable operation stop on single completed access round or on 5000 milliseconds timeout though sending setstoptrigger ASCII protocol command.
- Start of access operation is requested though sending lock ASCII protocol command.
- After completion of access operation previous start and stop trigger configurations are restored though sending setstarttrigger and setstoptrigger ASCII protocol commands.
- "Status Event" and "Read Event" notifications are disabled during the on-going access operation.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidLockTag (with Access Criteria)

Description

Request to perform lock operation for a particular tag with a particular RFID reader.

```
- (SRFID_RESULT) srfidLockTag:(int)readerID
aAccessCriteria:(srfidAccessCriteria*)accessCriteria
aAccessTagData:(srfidTagData**)accessTagData
aMemoryBank:(SRFID_MEMORYBANK)memoryBankID
aAccessPermissions:(SRFID_ACCESSPERMISSION)accessPermissions aPassword:(long)password
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(srfidAccessCriteria*)accessCriteria`

[in] Access criteria to identify the Tag on which the lock operation needs to be carried out by the SDK. Using the Access Criteria a tag can be chosen with one of the memory bank data.

`(srfidTagData**)accessTagData`

[out] Pointer to the srfidTagData object intended for storing results of the performed lock operation.

`(SRFID_MEMORYBANK)memoryBankID`

[in] Identifier of a particular memory bank on that lock operation shall be performed.

`(SRFID_ACCESSPERMISSION)accessPermissions`

[in] Lock action parameter.

`(long)password`

[in] Access password.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Operation has been performed successfully.

SRFID_RESULT_FAILURE

SDK has failed to perform operation.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

Invalid parameter (e.g. nil pointer).

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

(continued on next page)

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ ***NOTES***

- The caller is responsible for allocation, initialization and deallocation of srfidTagData object.
- Lock action parameter is applied to the specified by memoryBankID parameter memory bank or to kill and access password fields if memory bank is not specified.
- The operation is performed though a following set of ASCII protocol commands:
- Current start trigger configuration is requested though sending setstarttrigger ASCII protocol command with noexec parameter and is stored by the SDK.
- Current stop trigger configuration is requested though sending setstoptrigger ASCII protocol command with noexec parameter and is stored by the SDK.
- Criteria for access operation is configured in accordance with accessCriteria parameter though sending setaccessfilter ASCII protocol command.
- Start trigger configuration is updated to ignore hand-held trigger, enable operation start without any delay and disable repeat monitoring though sending setstarttrigger ASCII protocol command.
- Stop trigger configuration is updated to ignore hand-held trigger, enable operation stop on single completed access round or on 5000 milliseconds timeout though sending setstoptrigger ASCII protocol command.
- Start of access operation is requested though sending lock ASCII protocol command.
- After completion of access operation previous start and stop trigger configurations are restored though sending setstarttrigger and setstoptrigger ASCII protocol commands.
- "Status Event" and "Read Event" notifications are disabled during the on-going access operation.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidRequestBatteryStatus

Description

Request a particular RFID reader to provide new battery information related notification.

```
- (SRFID_RESULT) srfidRequestBatteryStatus:(int)readerID;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

Return Values

SRFID_RESULT_SUCCESS

The request has been sent to the RFID reader.

SRFID_RESULT_FAILURE

SDK has failed send a request to the RFID reader.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ ***NOTES***

- SDK causes a particular RFID reader to provide new battery information related notification via getdeviceinfo ASCII protocol command with .battery parameter.
- "Battery Event" notification will be provided when batter related information notification is received from the RFID reader.

srfidGetBatchModeConfig

Description

Request a particular RFID reader to provide batch mode configuration.

```
-- (SRFID_RESULT) srfidGetBatchModeConfig:(int)readerID
aBatchModeConfig:(SRFID_BATCHMODECONFIG*)batchModeConfig
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(SRFID_BATCHMODECONFIG*)batchModeConfig`

[out] Pointer to SRFID_BATCHMODECONFIG variable intended for storage of received batch mode configuration.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Batch mode configuration have been received successfully.

SRFID_RESULT_FAILURE

SDK has failed to receive batch mode configuration.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

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

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ ***NOTES***

- Information is requested through sending getattrinfo ASCII protocol command.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidSetBatchModeConfig

Description

Request to set batch mode configuration on a particular RFID reader.

```
- (SRFID_RESULT) srfidSetBatchModeConfig:(int)readerID
aBatchModeConfig:(SRFID_BATCHMODECONFIG)batchModeConfig
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(SRFID_BATCHMODECONFIG*)batchModeConfig`

[in] Required batch mode configuration.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Batch mode configuration have been applied successfully.

SRFID_RESULT_FAILURE

SDK has failed to apply batch mode configuration.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

Invalid parameter.

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

NOTES

- The applied configuration is lost after next power down. srfidSaveReaderConfiguration API shall be used to make the applied configuration persistent.
- Batch mode configuration is applied through sending setattr ASCII protocol command.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidGetTags

Description

Request to receive tags read in batch mode from a particular RFID reader.

```
-(SRFID_RESULT) srfidgetTags:(int)readerID aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Get tags of batch mode operation has been started successfully.

SRFID_RESULT_FAILURE

SDK has failed to read tags operation.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

Invalid parameters (e.g., an identifier of memory bank is not specified).

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

NOTES

- Read Inventory tags during batch mode is through sending get tags ASCII protocol command.
- If reportConfig and/or accessConfig parameter are not specified (e.g., are nil values) corresponding parameters of read ASCII protocol command will not be specified.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidGetConfigurations

Description

Request to get the reader configurations after batch mode reconnect.

```
- (SRFID_RESULT) srfidGetConfigurations;
```

Return Values

SRFID_RESULT_SUCCESS

The communication session has been established successfully after batch mode reconnect.

SRFID_RESULT_FAILURE

The communication session was not established.

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

srfidPurgeTags

Description

Request to purge tags read in batch mode from a particular RFID reader.

```
-(SRFID_RESULT) srfidPurgeTags:(int)readerID aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Purge tags of batch mode operation has been started successfully.

SRFID_RESULT_FAILURE

SDK has failed to purge tags operation.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

Invalid parameters (e.g., an identifier of memory bank is not specified).

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

✓ ***NOTES***

- Read Inventory tags during batch mode is removed through sending purge tags ASCII protocol command.
- If reportConfig and/or accessConfig parameter are not specified (e.g., are nil values) corresponding parameters of read ASCII protocol command will not be specified.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidBlockErase

Description

Request to perform block erase operation from a particular RFID reader.

```
-(SRFID_RESULT) srfidBlockErase:(int)readerID aTagID:(NSString *)tagID
aAccessTagData:(srfidTagData**)accessTagData
aMemoryBank:(SRFID_MEMORYBANK)memoryBankID aOffset:(short)offset aLength:(short)length
aPassword:(long)password aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(NSString*)tagID`

[in] Identifier of the tag on that the erase operation shall be performed.

`(srfidTagData**)accessTagData`

[out] Pointer to the srfidTagData object intended for storing results of the performed erase operation.

`(SRFID_MEMORYBANK)memoryBankID`

[in] Identifier of a particular memory bank on that erase operation shall be performed.

`(short)offset`

[in] Number of words offsetted from beginning of memory bank for where the erase operation shall be performed.

`(short)length`

[in] Number of words to erase.

`(long)password`

[in] Access password.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Block erase operation has been started successfully.

SRFID_RESULT_FAILURE

SDK has failed to perform block erase operation.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

Invalid parameters (e.g., an identifier of memory bank is not specified).

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

✓ ***NOTE*** If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidBlockErase (with Access Criteria)

Description

Request to perform block erase operation from a particular RFID reader.

```
- (SRFID_RESULT) srfidBlockErase:(int)readerID
aAccessCriteria:(srfidAccessCriteria*)accessCriteria
aAccessTagData:(srfidTagData**)accessTagData
aMemoryBank:(SRFID_MEMORYBANK)memoryBankID aOffset:(short)offset aLength:(short)length
aPassword:(long)password aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(srfidAccessCriteria*)accessCriteria`

[in] Access criteria to identify the Tag on which the block erase operation needs to be carried out by the SDK. Using the Access Criteria a tag can be chosen with one of the memory bank data.

`(srfidTagData**)accessTagData`

[out] Pointer to the srfidTagData object intended for storing results of the performed erase operation.

`(SRFID_MEMORYBANK)memoryBankID`

[in] Identifier of a particular memory bank on that erase operation shall be performed.

`(short)offset`

[in] Number of words offsetted from beginning of memory bank for where the erase operation shall be performed.

`(short)length`

[in] Number of words to erase.

`(long)password`

[in] Access password.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Block erase operation has been started successfully.

SSRFID_RESULT_FAILURE

SDK has failed to perform block erase operation.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

Invalid parameters (e.g., an identifier of memory bank is not specified).

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

✓ ***NOTE*** If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter..

srfidBlockPermaLock

Description

Request to perform block permalock operation from a particular RFID reader.

```
-(SRFID_RESULT) srfidBlockPermaLock:(int)readerID aTagID:(NSString *)tagID
aAccessTagData:(srfidTagData**)accessTagData
aMemoryBank:(SRFID_MEMORYBANK)memoryBankID aDoLock:(BOOL)doLock
aBlockPtr:(short)blockPtr aBlockRange:(short)blockRange aBlockMask:(NSString
*)blockMask aPassword:(long)password aStatusMessage:(NSString **)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(NSString*)tagID`

[in] Identifier of the tag on that the block perma lock shall be performed.

`(srfidTagData**)accessTagData`

[out] Pointer to the srfidTagData object intended for storing results of the performed block perma lock operation.

`(SRFID_MEMORYBANK)memoryBankID`

[in] Identifier of a particular memory bank on that block perma lock operation shall be performed.

`(BOOL)doLock`

[in] Specifies if permalock is performed.

`(short)blockPtr`

[in] Starting address of blockmask in units of 16 blocks.

`(short)blockRange`

[in] Mask range, in units of 16 blocks.

`(NSString*)blockMask`

[in] Bitmask representation of blocks to either perma lock (if bit asserted) or read current lock status (bit not asserted). Mandatory parameter, needs to be an ASCII Hex string.

`(long)password`

[in] Access password.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Block perma lock operation has been started successfully.

SRFID_RESULT_FAILURE

SDK has failed to perform block perma lock operation.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

Invalid parameters (e.g., an identifier of memory bank is not specified).

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

NOTES

- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.
- If the doLock parameter is set as 'false', permalockdata will be returned in RfidTagData.

srfidBlockPermaLock (with Access Criteria)

Description

Request to perform block permalock operation from a particular RFID reader.

```
- (SRFID_RESULT) srfidBlockPermaLock:(int)readerID
aAccessCriteria:(srfidAccessCriteria*)accessCriteria
aAccessTagData:(srfidTagData**)accessTagData
aMemoryBank:(SRFID_MEMORYBANK)memoryBankID aDoLock:(BOOL)doLock
aBlockPtr:(short)blockPtr aBlockRange:(short)blockRange aBlockMask:(NSString
*)blockMask aPassword:(long)password aStatusMessage:(NSString **)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(srfidAccessCriteria*)accessCriteria`

[in] Access criteria to identify the Tag on which the block permalock operation needs to be carried out by the SDK. Using the Access Criteria a tag can be chosen with one of the memory bank data.

`(srfidTagData**)accessTagData`

[out] Pointer to the srfidTagData object intended for storing results of the performed block perma lock operation.

`(SRFID_MEMORYBANK)memoryBankID`

[in] Identifier of a particular memory bank on that block perma lock operation shall be performed.

`(BOOL)doLock`

[in] Specifies if permalock is performed.

`(short)blockPtr`

[in] Starting address of blockmask in units of 16 blocks.

`(short)blockRange`

[in] Mask range, in units of 16 blocks.

`(NSString*)blockMask`

[in] Bitmask representation of blocks to either perma lock (if bit asserted) or read current lock status (bit not asserted). Mandatory parameter, needs to be an ASCII Hex string.

`(long)password`

[in] Access password.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Block perma lock operation has been started successfully.

SRFID_RESULT_FAILURE

SDK has failed to perform block perma lock operation.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

Invalid parameters (e.g. an identifier of memory bank is not specified).

(continued on next page)

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader

✓ **NOTES**

- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.
- If the doLock parameter is set as 'false', permalockdata will be returned in RfidTagData.

srfidGetAttribute

Description

Request to receive attributes from a particular RFID reader.

```
- (SRFID_RESULT) srfidGetAttribute:(int)readerID aAttrNum:(int)attrNum
aAttrInfo:(srfidAttribute**)attrInfo aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(srfidAttribute**)attrInfo`

[out] Pointer to srfidAttribute variable intended for storage of received attribute parameters.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

`(int)attrNum`

[in] Attribute number of a particular configuration.

Return Values

SRFID_RESULT_SUCCESS

Attributes have been received successfully.

SRFID_RESULT_FAILURE

SDK has failed to receive attributes.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

Invalid parameter (e.g. nil pointer).

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ **NOTES**

- Information is requested through sending getattrinfo ASCII protocol command.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

.srfidSetAttribute

Description

Request to set attribute values on a particular RFID reader.

```
- (SRFID_RESULT) srfidSetAttribute:(int)readerId attributeNumber:(int)attrNum
attributeValue:(int)attrVal attributeType:(NSString*)attrType
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(int)attrNum`

[in] Required attribute number.

`(int)attrVal`

[in] Required attribute value.

`(int)attrType`

[in] Required attribute type.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Reader configurations have been applied successfully.

SRFID_RESULT_FAILURE

SDK has failed to apply configuration.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

Invalid parameter.

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ ***NOTES***

- The applied configuration is lost after next power down. srfidSaveReaderConfiguration API shall be used to make the applied configuration persistent.
- Configuration is applied through sending setattr ASCII protocol command.
- If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

srfidSetAccessCommandOperationWaitTimeout

Description

Request to set the access command operation wait timeout (in milliseconds) for a particular RFID reader. The timeout value is applied for all access commands.

```
- (SRFID_RESULT) srfidSetAccessCommandOperationWaitTimeout:(int)readerId
aTimeoutMs:(int)timeoutMs
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(int)timeoutMs`

[in] Required timeout value (in milliseconds)

Return Values

SRFID_RESULT_SUCCESS

Reader configurations have been applied successfully.

SRFID_RESULT_FAILURE

SDK has failed to apply configuration.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

Invalid parameter.

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ ***NOTE*** The timeout value that is set applies to the current session with the reader. The timeout value is not persistent between sessions.

srfidLocateReader

Description

Request to locate a particular RFID reader.

```
- (SRFID_RESULT) srfidLocateReader:(int)readerID doEnabled:(BOOL)doEnabled
aStatusMessage:(NSString**)statusMessage;
```

Parameters

`(int)readerID`

[in] Unique identifier of a particular RFID reader assigned by SDK.

`(BOOL)doEnabled`

[in] Specifies whether the locate reader operation is to be performed or stopped.

`(NSString**)statusMessage`

[out] Pointer to NSString variable intended for storage of status message if an error has been reported by the RFID reader via ASCII interface.

Return Values

SRFID_RESULT_SUCCESS

Attributes have been received successfully.

SRFID_RESULT_FAILURE

SDK has failed to receive attributes.

SRFID_READER_NOT_AVAILABLE

The request was not processed because the RFID reader specified by readerID parameter was not active or available.

SRFID_RESULT_INVALID_PARAMS

Invalid parameter (e.g. nil pointer).

SRFID_RESULT_RESPONSE_ERROR

An error has been reported by the RFID reader via ASCII interface.

SRFID_RESULT_RESPONSE_TIMEOUT

Timeout has occurred while waiting for a response from the RFID reader.

SRFID_RESULT_ASCII_CONNECTION_REQUIRED

The request was not processed because an ASCII protocol level connection with the RFID reader specified by readerID parameter has not been established.

✓ ***NOTE*** If an error has been reported by the RFID reader the received error message is stored in statusMessage parameter.

Notifications

SDK callback interface is defined by srfidISdkApiDelegate Objective C protocol. Registration of a particular object which conforms to srfidISdkApiDelegate protocol is required to receive particular notifications from Zebra RFID SDK for iOS.

srfidEventReaderAppeared

Description

"Device Arrival" notification informs about appearance of a particular available RFID reader.

```
- (void)srfidEventReaderAppeared:(srfidReaderInfo*)availableReader;
```

Parameters

`(srfidReaderInfo*)availableReader`

srfidReaderInfo object representing an appeared RFID reader.

✓ ***NOTE*** The SDK is responsible for allocation and deallocation of availableReader object. The SDK will deallocate availableReader object immediately after execution of the callback.

srfidEventReaderDisappeared

Description

"Device Disappeared" notification informs about disappearance of a particular available RFID reader.

```
- (void)srfidEventReaderDisappeared:(int)readerID;
```

Parameters

`(int)readerID`

Unique identifier of a disappeared available RFID reader assigned by SDK.

srfidEventCommunicationSessionEstablished

Description

"Session Established" notification informs about appearance of a particular active RFID reader.

```
- (void)srfidEventCommunicationSessionEstablished:(srfidReaderInfo*)activeReader;
```

Parameters

`(srfidReaderInfo*)activeReader`

srfidReaderInfo object representing an appeared active RFID reader.

✓ ***NOTE*** The SDK is responsible for allocation and deallocation of activeReader object. The SDK will deallocate activeReader object immediately after execution of the callback.

srfidEventCommunicationSessionTerminated

Description

"Session Terminated" notification informs about disappearance of a particular active RFID reader.

```
- (void)srfidEventCommunicationSessionTerminated:(int)readerID
```

Parameters

`(int)readerID`

Unique identifier of a disappeared active RFID reader assigned by SDK.

srfidEventReadNotify

Description

"Read Event" notification informs about reception of a tag related data from a particular active RFID reader during on-going rapid read, inventory or access operation.

```
- (void)srfidEventReadNotify:(int)readerID aTagData:(srfidTagData*)tagData;
```

Parameters

`(int)readerID`

Unique identifier of a particular active RFID reader assigned by SDK.

`(srfidTagData*)tagData`

srfidTagData* object representing tag data received from RFID reader.

NOTES

- The SDK is responsible for allocation and deallocation of tagData object. The SDK will deallocate tagData object immediately after execution of the callback.
- The callback is always executed from one of SDK background threads.

srfidEventStatusNotify

Description

"Status Event" notification is triggered when one of operation status notifications if received from a particular active RFID reader.

```
- (void)srfidEventStatusNotify:(int)readerID aEvent:(SRFID_EVENT_STATUS)event
aNotification:(id)notificationData;
```

Parameters

`(SRFID_EVENT_STATUS)event`

Identifier of received status notification.

`(int)readerID`

Unique identifier of a particular active RFID reader assigned by SDK.

`(id)notificationData`

Notification object of received status notification.

NOTE The callback is always executed from one of SDK background threads.

srfidEventProximityNotify

Description

"Proximity Event" notification informs about reception of a tag proximity related data from a particular active RFID reader during on-going tag locationing operation.

```
- (void)srfidEventProximityNotify:(int)readerID
aProximityPercent:(int)proximityPercent;
```

Parameters

`(int)readerID`

Unique identifier of a particular active RFID reader assigned by SDK.

`(int)proximityPercent`

Tag proximity value in percents.

NOTE The callback is always executed from one of SDK background threads.

srfidEventTriggerNotify

Description

"Trigger Event" notification informs about reception of a particular notification related to pressing/releasing of a hand-held trigger from a particular active RFID reader.

```
- (void)srfidEventTriggerNotify:(int)readerID
aTriggerEvent:(SRFID_TRIGGEREVENT)triggerEvent;
```

Parameters

`(int)readerID`

Unique identifier of a particular active RFID reader assigned by SDK.

`(SRFID_TRIGGEREVENT)triggerEvent`

Type of the reported trigger event.

✓ ***NOTE*** The callback is always executed from one of SDK background threads.

srfidEventBatteryNotify

Description

"Battery Event" notification informs about reception of a particular notification related to battery information from a particular active RFID reader.

```
- (void)srfidEventBatteryNotity:(int)readerID
aBatteryEvent:(srfidBatteryEvent*)batteryEvent;;
```

Parameters

`(int)readerID`

Unique identifier of a particular active RFID reader assigned by SDK.

`(srfidBatteryEvent*)batteryEvent`

Battery related information represented by srfidBatteryEvent object.

✓ ***NOTE*** The callback is always executed from one of SDK background threads.

Chapter 4 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

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 supports simultaneous interaction with multiple active RFID readers. To distinguish various RFID readers the SDK assigns the unique integer identifier for each RFID reader when it becomes available first time.

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.

```
/* 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]);

    /* 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;

/* 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
```

(continued on next page)

```
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;
```

(continued on next page)

```
     /* 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 {
        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];
```

(continued on next page)

```
     /* 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];

     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]);

     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;
     }
```

(continued on next page)

```
        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];

        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 {
```

(continued on next page)

```
     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];

/* 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");
}
```

(continued on next page)

```
/* 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 {
     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;
```

(continued on next page)

```
/* 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;
     }
}

[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]);

     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;
         }

         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];
```

(continued on next page)

```
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.

```
/* 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];
```

(continued on next page)

```
     /* 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");
     }
     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];
```

(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");
     }
} 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 {
     /* 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];

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");
}

/* 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
```

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.

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)
```

(continued on next page)

```
     { 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];
```

Chapter 5 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 1-3*).
- 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>.

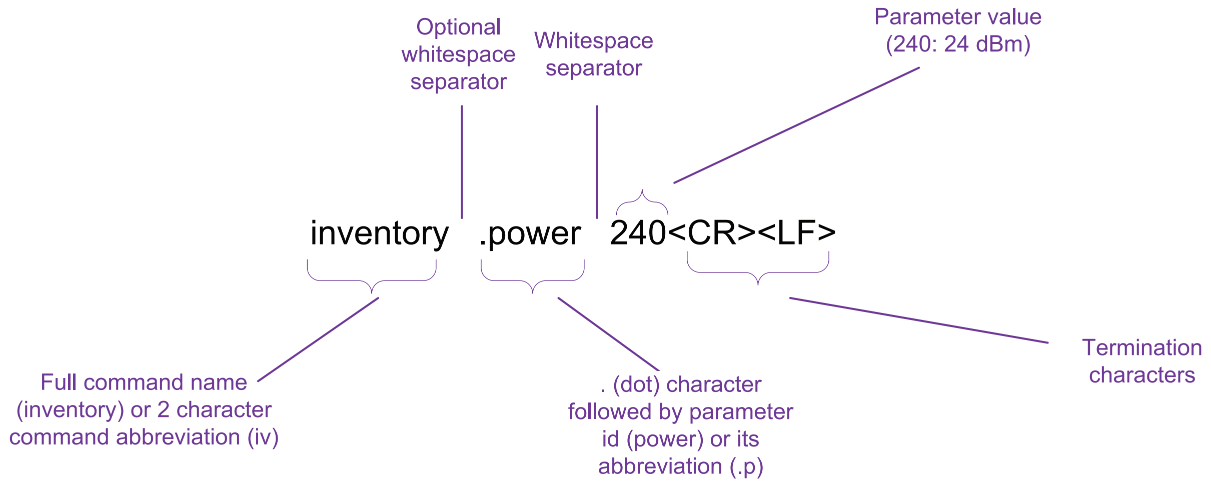

Optional whitespace separator
Whitespace separator
Parameter value (240: 24 dBm)
inventory .power 240<CR><LF>
Full command name (inventory) or 2 character command abbreviation (iv)
. (dot) character followed by parameter id (power) or its abbreviation (.p)
Termination characters

Figure 5-1 *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 5-1 on page 5-2*:

```
Command:inventory,Status:0,EPC:,RSSI:,TS:<CR><LF>
,,3200112233445566778889901,-45,22334455<CR><LF>
,,3200112233445566778889912,-56,<CR><LF>
,,3200112233445566778889903,-60,<CR><LF>
,,3200112233445566778889904,-44,22334465<CR><LF>
<CR><LF>
```

Figure 5-2 illustrates the components of the response for a command.

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

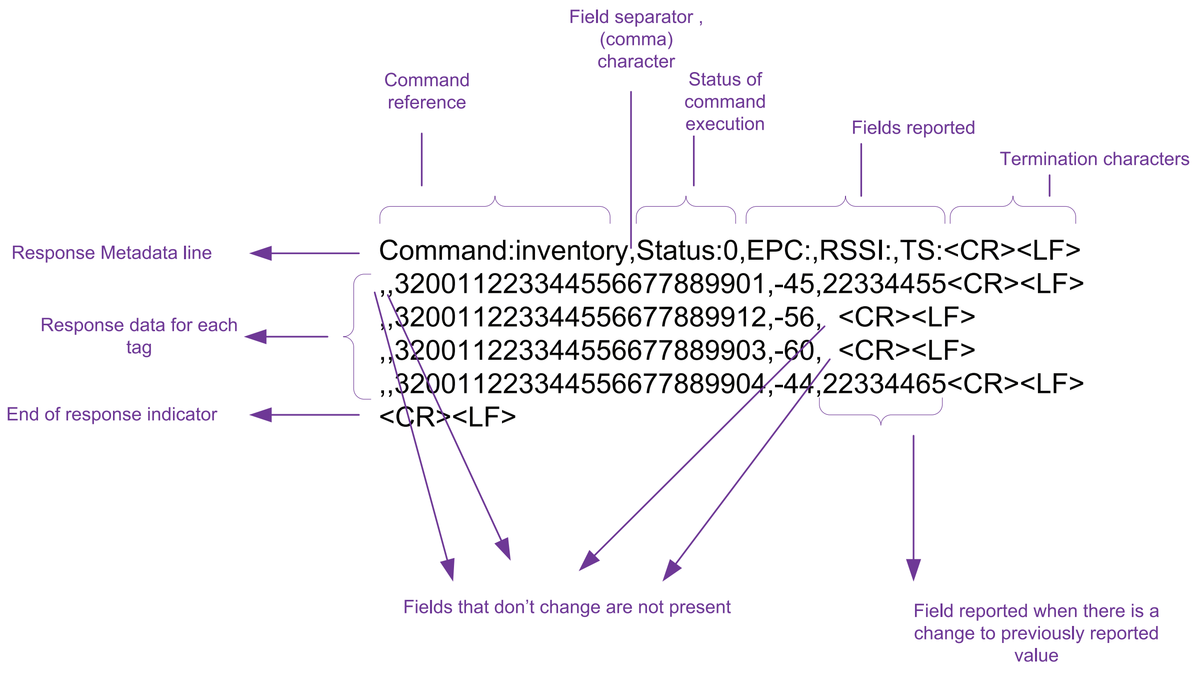
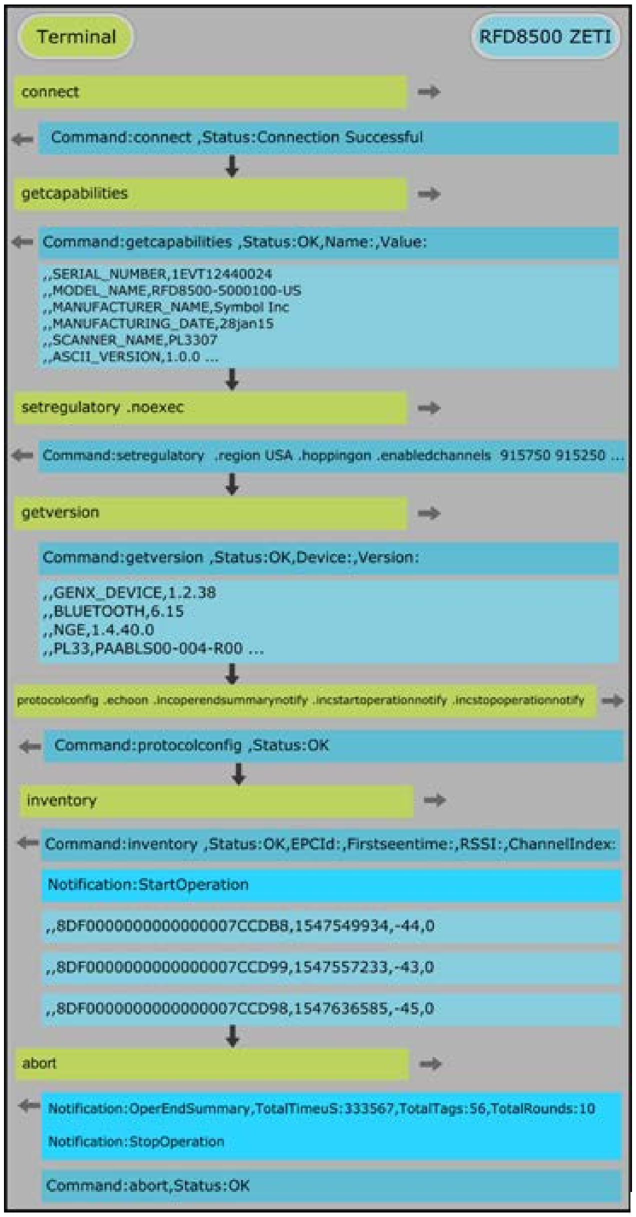

Field separator , (comma) character
Command reference
Status of command execution
Fields reported
Termination characters
Response Metadata line
Command:inventory,Status:0,EPC:,RSSI:,TS:<CR><LF>
,,320011223344556677889901,-45,22334455<CR><LF>
,,320011223344556677889912,-56, <CR><LF>
,,320011223344556677889903,-60, <CR><LF>
,,320011223344556677889904,-44,22334465<CR><LF>
<CR><LF>
Response data for each tag
End of response indicator
Fields that don't change are not present
Field reported when there is a change to previously reported value

Figure 5-2 *Components of the Response Command*

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

General Flow.



```
connect
Command:connect ,Status:Connection Successful
getcapabilities
Command:getcapabilities ,Status:OK,Name:,Value:
,,SERIAL_NUMBER,1EVT12440024
,,MODEL_NAME,RFD8500-5000100-US
,,MANUFACTURER_NAME,Symbol Inc
,,MANUFACTURING_DATE,28jan15
,,SCANNER_NAME,PL3307
,,ASCII_VERSION,1.0.0 ...
setregulatory .noexec
Command:setregulatory  .region USA .hoppingon .enabledchannels  915750 915250 ...
getversion
Command:getversion ,Status:OK,Device:,Version:
,,GENX_DEVICE,1.2.38
,,BLUETOOTH,6.15
,,NGE,1.4.40.0
,,PL33,PAABLS00-004-R00 ...
protocolconfig .echoon .incoperendsummarynotify .incstartoperationnotify .incstopoperationnotify
Command:protocolconfig ,Status:OK
inventory
Command:inventory ,Status:OK,EPCId:,Firstseentime:,RSSI:,ChannelIndex:
Notification:StartOperation
,,8DF0000000000000007CCDB8,1547549934,-44,0
,,8DF0000000000000007CCD99,1547557233,-43,0
,,8DF0000000000000007CCD98,1547636585,-45,0
abort
Notification:OperEndSummary,TotalTimeuS:333567,TotalTags:56,TotalRounds:10
Notification:StopOperation
Command:abort,Status:OK
```

Figure 5-3 *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
```

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,,00000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000
```

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:
,,E2002849491502351020B318,1912599493,-32,,2
lo .w AABBCCDD .usermem 2
Command:lock ,Status:OK,EPCId:,Firstseentime:,RSSI:,lockStatus:
,,E2002849491502351020B318,1942830592,-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:
,,E2002849491502351020B318,1967721994,-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:
,,E2002849491502351020B318,1977931276,-30,,2
rd
Command:read ,Status:OK,EPCId:,Firstseentime:,RSSI:,readStatus:,user:
,,E2002849491502351020B318,1984136665,-30,,11223344000000000000000000000000000000
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:
,,8DF000000000000000812E39,2279812125,-35,,1122334400000000
kl .w 11223344
Command:kill ,Status:OK,EPCId:,Firstseentime:,RSSI:,killStatus:
,,8DF000000000000000812E39,2298543322,-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:
,,000000000000000000000253,859739100,-37
,,8DF000000000000000812E3A,864135617,-36
Notification:OperEndSummary,TotalTimeuS:51742,TotalTags:2,TotalRounds:2
,,000000000000000000000253,864162904,-37
,,8DF000000000000000812E3B,865156101,-36
Notification:OperEndSummary,TotalTimeuS:62170,TotalTags:2,TotalRounds:2
,,000000000000000000000252,865161345,-37
,,8DF000000000000000812E3B,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
,,000000000000000000000252,1092359947,-38
,,000000000000000000000253,1341606260,-37
,,000000000000000000000252,1341610687,-38
,,0000000000000000000000AD,1341615370,-45
,,000000000000000000000252,1341622204,-38
,,000000000000000000000253,1341628860,-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:
,,E2002849491502361020B31C,1031167844,-32,,E2002849491502361020B31C
```

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:
,,E2002849491502361020B31C,5758908545,-30,,00000000,,2,,,AABBCCDD
,,E2002849491502361020B31C,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 0000965644402375796C69664 .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
```

(continued on next page)

```
Notification:StopOperation
```

Use Untraceable command to unhide EPC memory bank.

```
uc .el 6 .ci 1
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 8DF0000000000000812E3B
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:
,,0000000000000000000000AD,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
,,0000000000000000000000AD,341390219,-30,27696
,,8DF000000000000000812E54,341394089,-35,21719
,,8DF000000000000000812E42,341397957,-30,6255
,,8DF000000000000000812E5B,341401829,-32,35067
```

(continued on next page)

```
,,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-44* for the possible errors reported back for ZETI commands.

Table A-1 *ZETI Interface Command Reference*

Command	Parameters		Response	Example
	Parameter ID	Options, Default Value, Range		
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: Parameter ID	Parameters: Options, Default Value, Range	Response	Example
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-43*.		
	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		Response	Example
	Parameter ID	Options, Default Value, Range		
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 *srfidGetSupportedLinkProfiles 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: Parameter ID	Parameters: Options, Default Value, Range	Response	Example

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)*

	Parameters			
Command	**Parameter ID**	**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>`
				`,,400812345678098765430 01D,4150045406,4150045406,3000,-59,0,0,1 <CR><LF>`
				`,,1111CCCCBBBBAAAA00009999,4150719034,4150719034,3000,-61,0,0,1 <CR><LF>`
				`,,8DF000000000000000812E9A,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)*

	Parameters			
Command	**Parameter ID**	**Options, Default Value, Range**	**Response**	**Example**
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>`
				`,,320011223344556677889901,12 3456,-40<CR><LF>`
				`,,320011223344556677889912,22 3457,-44<CR><LF>`
				`,,320011223344556677889903,32 3454,-42<CR><LF>`
				`,,320011223344556677889904,42 3458,-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		Response	Example
	Parameter ID	Options, Default Value, Range		

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	Parameters: 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)*

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

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

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

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:`
				`,,112222223333444400000256,30`
				`2225212,-44,1,,12753000,,E200`
				`6004`
				`,,2F2203447334C3100002EB80,30`
				`2240488,-44,1,,AFC63000,,E200`
				`3412`
				`,,2F2203447334C3100002EB80,30`
				`2256823,-44,1,,AFC63000,,E200`
				`3412`
				`,,112222223333444400000256,30`
				`2272689,-45,1,,12753000,,E200`
				`6004`
				`,,2F2203447334C3100002EB80,30`
				`2291156,-44,1,,AFC63000,,E200`
				`3412`
				`,,112222223333444400000256,30`
				`2303905,-45,1,,12753000,,E200`
				`6004`
				`,,2F2203447334C3100002EB80,30`
				`2326571,-44,1,,AFC63000,,E200`
				`3412`
				`,,112222223333444400000256,30`
				`2339173,-44,1,,12753000,,E200`
				`6004`
				`,,2F2203447334C3100002EB80,30`
				`2355104,-44,1,,AFC63000,,E200`
				`3412`
				`,,112222223333444400000256,30`
				`2368679,-45,1,,12753000,,E200`
				`6004`
				`,,112222223333444400000256,30`
				`2388045,-45,1,,12753000,,E200`
				`6004`
				`,,2F2203447334C3100002EB80,30`
				`2400630,-44,1,,AFC63000,,E200`
				`3412`
				`,,112222223333444400000256,30`
				`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 000000000000000000000002A0 .epm 000000000000000000000FF0<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>`
				`,,0000000000000000000000AD,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		Response	Example
	Parameter ID	Options, Default Value, Range		
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		Response	Example
	Parameter ID	Options, Default Value, Range		

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)*

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

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	Parameters: 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: Parameter ID	Parameters: Options, Default Value, Range	Response	Example
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)*

	Parameters			
Command	**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:getcapabilit ies ,Status:OK,Name:,Value:<CR><L F> ,,SERIAL_NUMBER,720001EVT1244 003<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>`
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,Cau se:User Request,Level:90,Charging:tru e <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)			

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

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

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		Response	Example
	Parameter ID	Options, Default Value, Range		
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		Response	Example
	Parameter ID	Options, Default Value, Range		
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,-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.		
	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)*

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

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		Response	Example
	Parameter ID	Options, Default Value, Range		
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		Response	Example
	Parameter ID	Options, Default Value, Range		
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>`

Table A-2 *Possible Select Action Values*

Action	Matching	Non-matching
0 (default)	Assert SL or inventoried $\longrightarrow$ *A*	De-assert SL or inventoried $\longrightarrow$ *B*
1	Assert SL or inventoried $\longrightarrow$ *A*	Do nothing
2	Do nothing	De-assert SL or inventoried $\longrightarrow$ *B*
3	Negate SL or (A $\longrightarrow$ B, B $\longrightarrow$ A)	Do nothing
4	De-assert SL or inventoried $\longrightarrow$ *B*	Assert SL or inventoried $\longrightarrow$ *A*
5	De-assert SL or inventoried $\longrightarrow$ *B*	Do nothing
6	Do nothing	Assert SL or inventoried $\longrightarrow$ *A*
7	Do nothing	Negate SL or (A $\longrightarrow$ B, B $\longrightarrow$ 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`

authenticate(au)	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
	msglen(l)	Range Validation and "Authenticate Message Mismatch"
	csi(i)	Range Validation
	resplength(rp)	Range Validation
	msgdata(md)	Range Validation
	power(p)	Range Validation

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`

Table A-3 *ZETI Interface Command Errors (Continued)*

Command	Parameter ID	Possible Errors
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(eg. 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:,re`
		`adStatus:,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 we can get TARI validation error as Follows:
		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 LinkProfile We can Get:
		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.

ZEBRA

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

MN002222A02 Revision A - March 2016