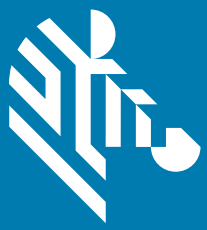
SMARTDEX



ZEBRA

Integration Guide for Android

MN-002802-02

Copyright

Terms of Use

- Proprietary Statement

  This manual contains proprietary information of Zebra Technologies Corporation and its subsidiaries (“Zebra Technologies”). It is intended solely for the information and use of parties operating and maintaining the equipment described herein. Such proprietary information may not be used, reproduced, or disclosed to any other parties for any other purpose without the express, written permission of Zebra Technologies.
- Product Improvements

  Continuous improvement of products is a policy of Zebra Technologies. All specifications and designs are subject to change without notice.
- Liability Disclaimer

  Zebra Technologies takes steps to ensure that its published Engineering specifications and manuals are correct; however, errors do occur. Zebra Technologies reserves the right to correct any such errors and disclaims liability resulting therefrom.
- Limitation of Liability

  In no event shall Zebra Technologies or anyone else involved in the creation, production, or delivery of the accompanying product (including hardware and software) be liable for any damages whatsoever (including, without limitation, consequential damages including loss of business profits, business interruption, or loss of business information) arising out of the use of, the results of use of, or inability to use such product, even if Zebra Technologies has been advised of the possibility of such damages. Some jurisdictions do not allow the exclusion or limitation of incidental or consequential damages, so the above limitation or exclusion may not apply to you.

Revision History

Changes to the original guide are listed below:

Change	Date	Description
-01 Rev A	8/2016	draft
-02 Rev A	3/2018	New screen captures. Updates to procedures.

Table of Contents

About This Guide

Introduction

Direct EXchange

Direct EXchange (DEX) is a digital communication protocol that extends the UCS (Uniform Communication Standard) that enables direct store delivery drivers to transmit digital invoices to the retailer's receiving clerk at the loading dock.

Created by the Uniform Code Council (UCC), DEX has been adopted by most national grocery chains. This standardized system reduces time, costs and inaccuracies inherent in paper invoices. Originally designed in the early 1980's, the DEX protocol has been revised several times over the years. The most current version is 5030 version, released in 2006. Unfortunately, adoption of protocol upgrades has been un-even in the marketplace, requiring DEX software to be adaptable to the specific requirements of each retailer served.

SmartDEX

SmartDEX solution is a hardware and software bundle combining Zebra Android handheld mobile computers and embedded DEX software from Versatile Mobile Systems.

Since the late 1990's Versatile Mobile Systems has offered DEX software solutions on every major mobile operating system, including DOS, Pocket PC, Windows Mobile, and now, Android.

SmartDEX on Android is currently available only as an embedded client. This means that SmartDEX must be called from a parent route accounting or delivery application. That application must supply the necessary invoice data to SmartDEX in order to run a DEX session.

In order to cover all versions of the DEX servers used by retailers today, SmartDEX is compliant with all UCS protocols from version 4010 to 5030.

All versions of SmartDEX use two types of data:

- **Route Data** - Route Data contains all information and invoices for a single route. The data is stored as a text file with simple formatting that can be imported and/or exported by a large number of programs. Your system will need to create the route (invoice) data in this format for the DEX Mobile Client. (see DEX Client Data Format).
- **Results Data** - The Results Data is generated by the Mobile Client. This data contains the delivery notification and adjustments for DEX sessions performed using the mobile device. This data can be easily imported by your handheld application if detailed histories of adjustments and results of DEX sessions are required. (see Results Data Format).

Chapter Descriptions

Topics covered in this guide are as follows:

- Embedded Installation describes how to install the SmartDEX application on an Android device when used with a line of business application.
- Demo Installation describes how to install SmartDEX on a mobile computer for demonstration and testing.
- Configuration Settings provides instructions on configuring the SmartDEX settings.
- Results Data provides an example of the results data file.
- DEX Client Data Format describes the format for the DEX Client Data.
- Results Data Format describes the format for the Result Data.
- SmartDEX Licensing describes the SmartDEX licensing information.
- Implementation described embedded version implementation.

Notational Conventions

The following conventions are used in this document:

- **Bold** text is used to highlight the following:
  - Dialog box, window and screen names
  - Drop-down list and list box names
  - Check box and radio button names
  - Icons on a screen
  - Key names on a keypad
  - Button names on a screen.
- Bullets (•) indicate:
  - Action items
  - Lists of alternatives
  - Lists of required steps that are not necessarily sequential.
- Sequential lists (e.g., those that describe step-by-step procedures) appear as numbered lists.

Related Documents and Software

The following documents provide more information about the SmartDEX application.

- SmartDEX Use Guide, p/n MN-002458-xx
- DEX Scan & Pair User Guide, p/n MN-002803-xx

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

Service Information

If you have a problem with your equipment, contact Zebra Global Customer Support for your region. Contact information is available at: http://www.zebra.com/support.

When contacting support, please have the following information available:

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

Zebra responds to calls by email or telephone within the time limits set forth in support agreements.

If your problem cannot be solved by Zebra Customer 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 Zebra business product from a Zebra business partner, contact that business partner for support.

Embedded Installation

Introduction

This chapter describes how to install the SmartDEX application on an Android device when used with a line of business application.

Installing SmartDEX

To install the SmartDEX application:

1. Download the SmartDEX application from the Zebra Support Central web site, http://www.zebra.com/support.
2. Uncompress the file into a folder on the host computer.
3. Ensure that the mobile computer is set to allow installation of applications from external source. See system administrator for more information.
4. Connect a USB cable to the device and host computer.
5. On the mobile computer, open the notification panel and touch **USB for charging**. In the dialog box, select **File transfer**.
6. On the host computer, open a file explorer application and locate the SmartDEX files.
7. In another file explorer window, open the mobile device directory tree.
8. Open **Internal Storage** > **Download** folder.
9. Copy the **`SmartDEX.apk`** file into the **`Download`** folder.
10. Disconnect the USB cable from the device.
11. On the mobile computer, open the All Apps screen.
12. Touch **File Browser**.
13. Touch **Internal** folder.
14. Touch **Download** folder.

Figure 1 File Browser

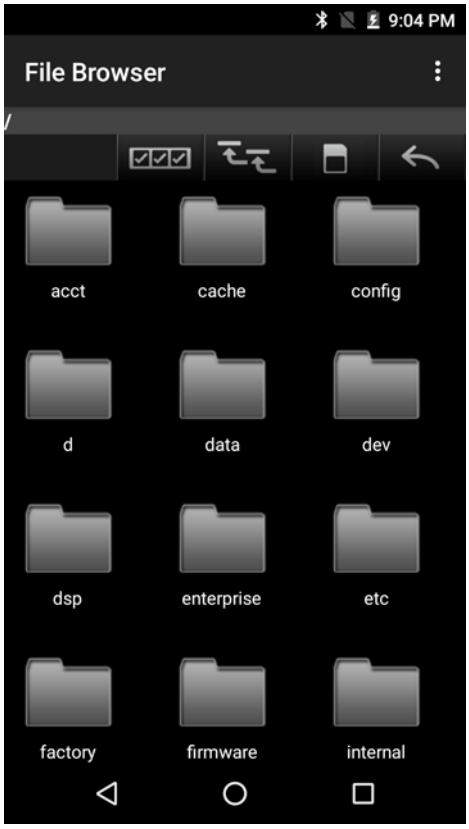
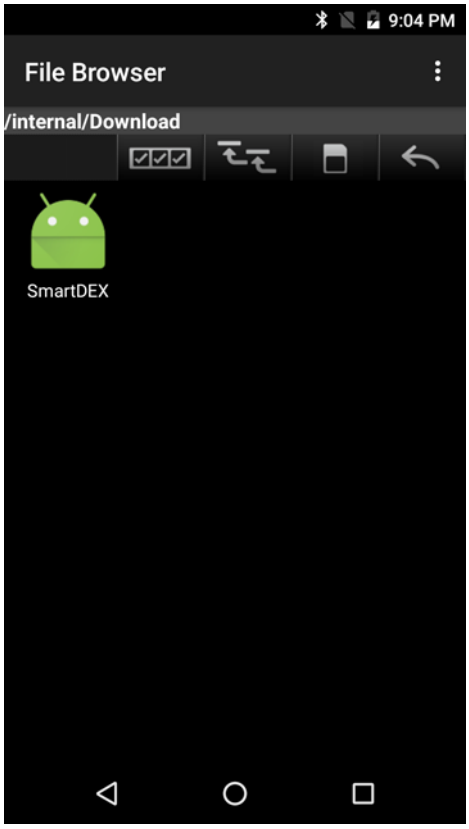
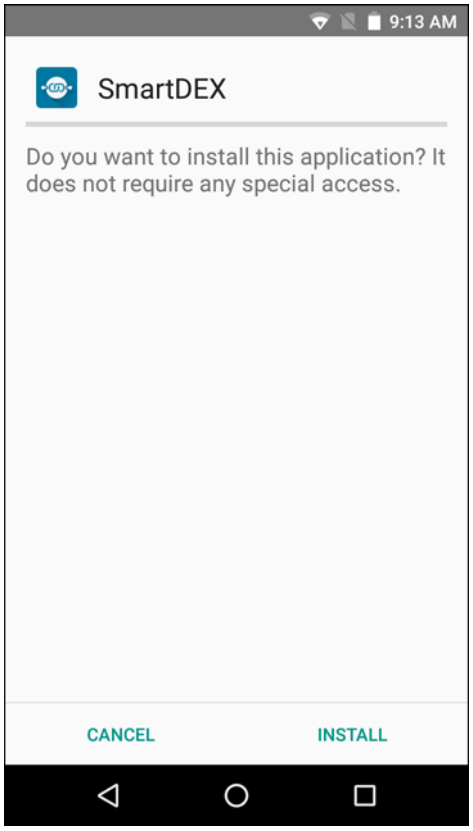


Figure 2 APK File



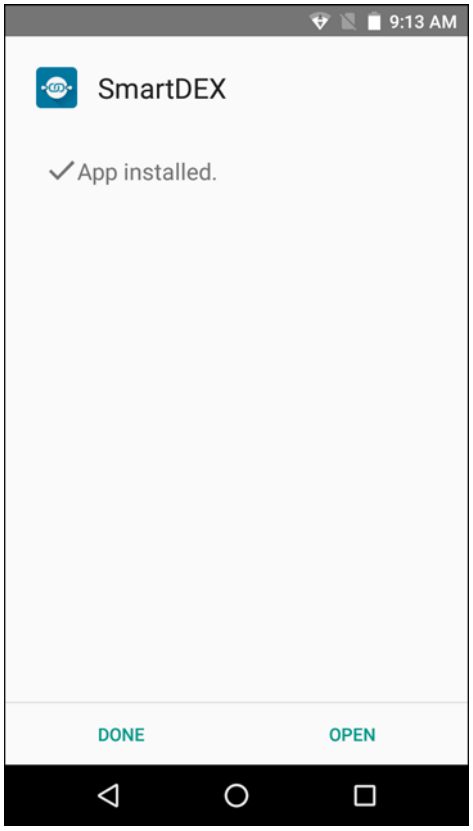
15. Touch the **SmartDEX.apk** file to begin the installation process.

Figure 3 Install Screen



16. Touch **Install**.
17. When the installation is complete, a verification message appears.

Figure 4 Verification Screen



18. Touch **Open** to launch the application or **Done** to close the installation window.

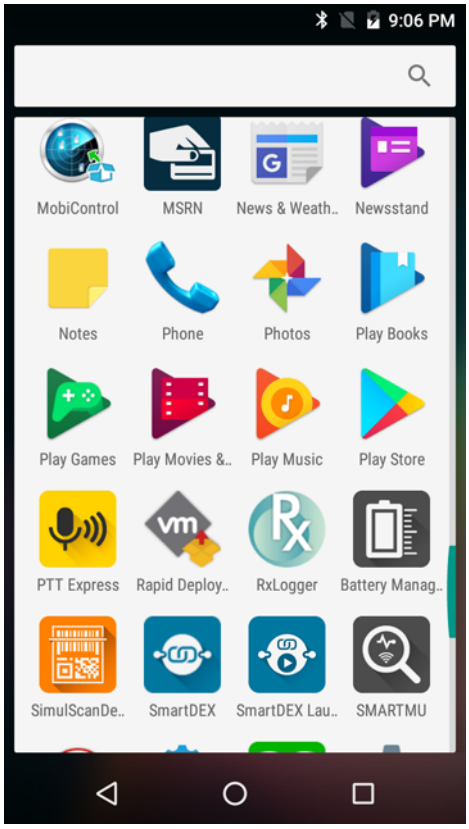
Registration

After the SmartDEX client has been successfully installed onto the mobile computer, the client license must be registered before using the software. Registration of the client license requires valid account login credentials which are provided by Zebra.

NOTE: The Embedded version of the SmartDEX client is intended to be started by another program (the application). However, to simplify the registration process, it may be started directly (manually).

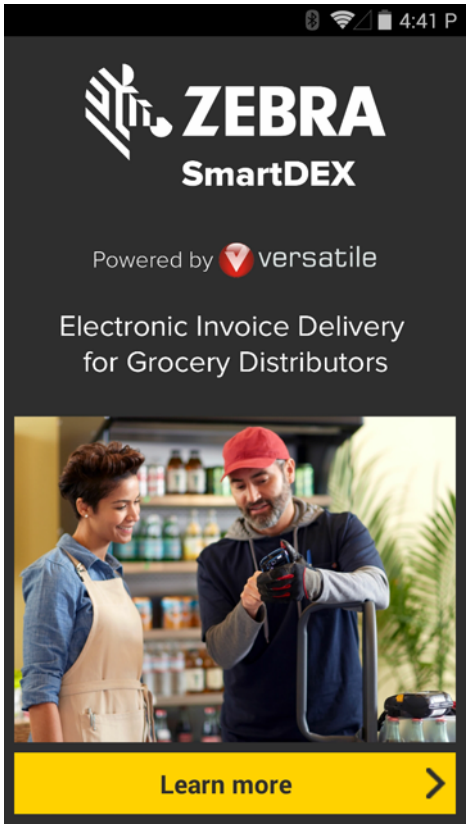
1. Open All Apps screen.

Figure 5 Applications Screen



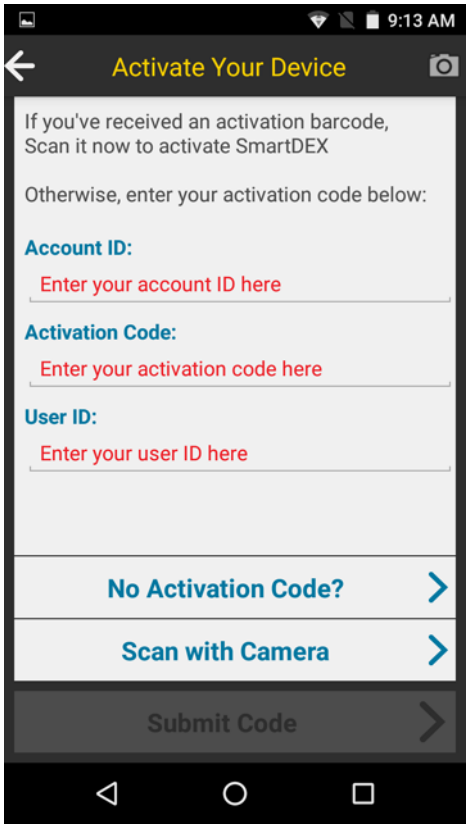
2. Touch the **SmartDEX** icon. SmartDEX opens in marketing mode.

Figure 6 Informational Screen



3. Touch **Learn More**.
4. Touch **Activate SmartDEX**. The **Active Your Device** screen appears.

Figure 7 Activate Your Device Screen



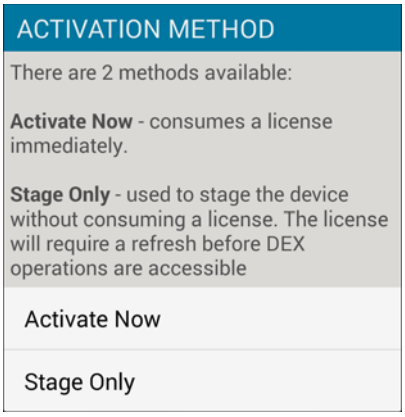
NOTE: The **Account ID** and **Activation Code** are obtained by the agreement with Zebra. If Zebra provided an activation QR code, scan now to fill the **Account ID** and **Activation Code** fields.

5. In the **Account ID** text field, enter the account ID (case sensitive).
6. In the **Activation Code** text field, enter the activation code (not case sensitive).

NOTE: The **User ID** can be anything useful to help identify the user of the device.

7. In the **User ID** text field, enter a user ID.
8. Touch **Submit Code**. The **Activation Method** dialog box appears.

Figure 8 Activation Method



9. If using the device for testing, integration, or providing to a user for live use, select **Activate Now**.

   If staging a number of devices for distribution to end users, choose **Stage Only**. Stage Only allows IT staff to prepare all devices for DEX use, without immediately activating licenses. When the end users starts SmartDEX for the first time, it activates the license at that time.

   The registration process takes just a few seconds. Ensure that the device has been configured and enabled to access the internet before attempting to register.

Configuration Settings

Introduction

This chapter provides instructions on configuring the SmartDEX settings.

NOTE: Settings may also be accessed from the main screen by touching the padlock icon in the upper menu bar area.

Settings

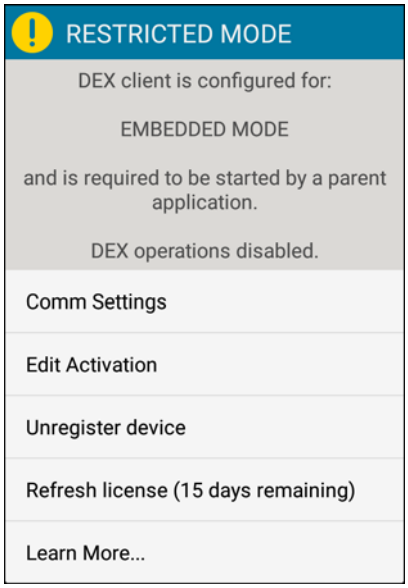
To configure the settings:

1. From the **Apps** screen, touch the **SmartDEX** icon.
2. Touch **Settings**.

NOTE: If the client is currently registered, an **Restricted Mode** message displays. There will only be a choice between **Go To Comm Settings**, **Unregister Device** or **Refresh license (x days remaining)**.

To exit without choosing an option, touch the device Back button to dismiss the dialog box and exit SmartDEX application.

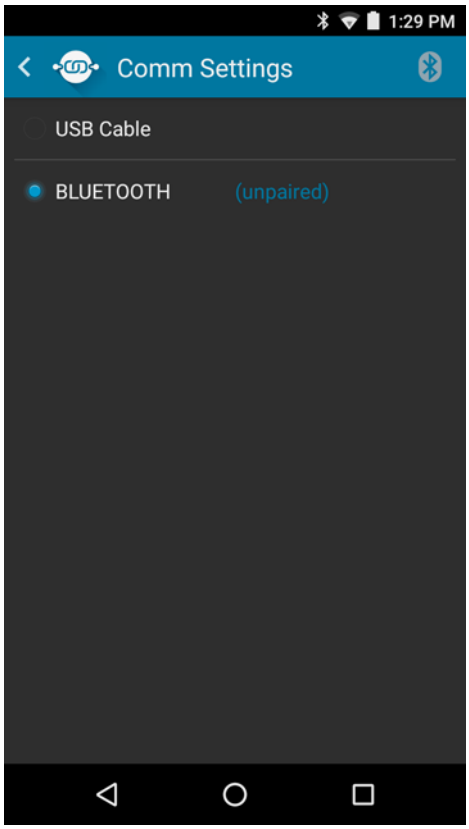
Figure 9 Restricted Mode Message



Touch **Goto Comm Settings**.

The **Comm Settings** screen allows the user to choose the interface method for connecting to the retailer's DEX server.

Figure 10 SmartDEX Comm Settings



Radio buttons allow the user to select either a **USB Cable** or the **Bluetooth** communications.

The following devices are certified to work with SmartDEX.

Table 1 Certified Devices

Connection	Device
USB Cable	CBL-TC7X-DEX1-01 (TC70x only)
Bluetooth	DX30 Bluetooth Key Fob

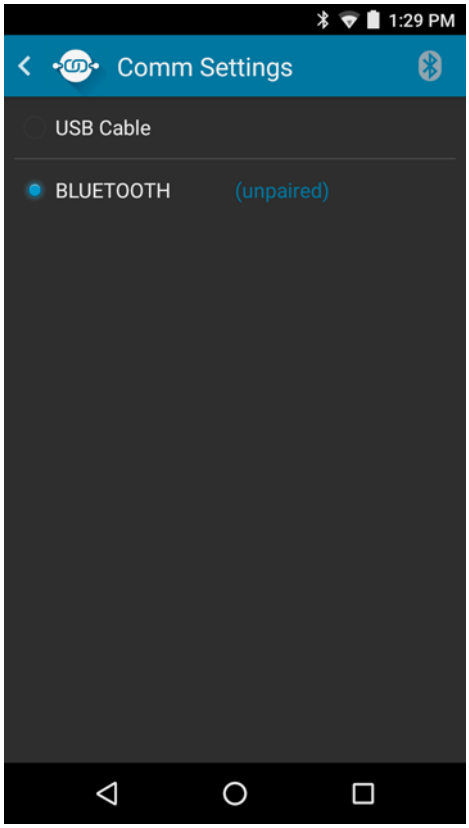
Pairing to the DX30 Bluetooth Key Fob

The status of the DX30 is indicated by **Un-paired** if it is un-paired, or by the **DX30 serial number** if it is paired.

There are three Bluetooth pairing methods.

- Scan to Pair
- Tap to Pair
- Discovery.

Figure 11 Comm Settings Screen



1. Touch the Bluetooth icon.
2. Select one of the setting options.
   - **Scan to Pair**. See Scan to Pair on page 20 for the next steps.
   - **Tap to Pair**. See Tap to Pair on page 21 for the next steps.

- **Discovery**. See Discover to Pair on page 23 for the next steps.

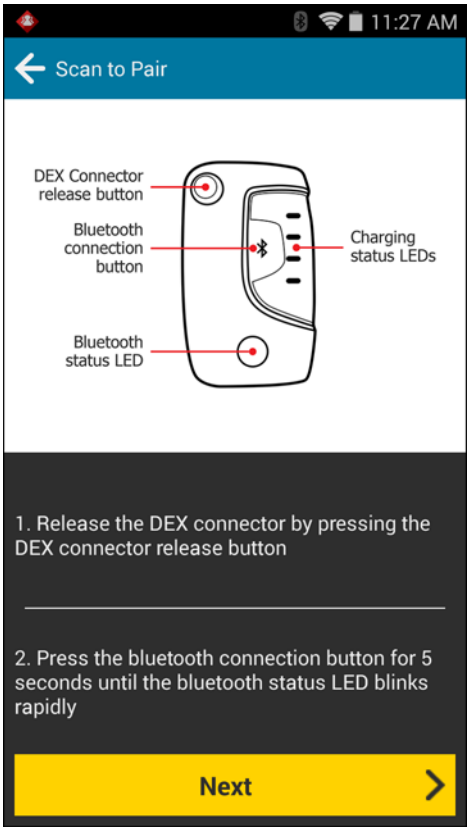
Scan to Pair

To scan and pair:

1. Touch **Scan to Pair.**

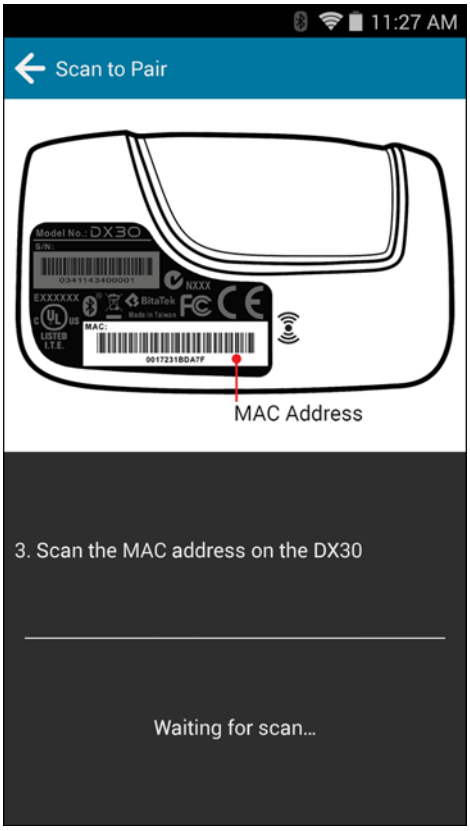
   The **Scan to Pair** screen appears instructing the user to prepare the DX30 for pairing. An image of the bar code location is displays for clarity.

Figure 12 Scan & Pair Screen



2. On the DX30, press the release button to open the DEX connector.
3. On the DX30, hold the pairing button down for five seconds. The blue light flashes for a second, stops flashing, and then continues to flash indicating the DX30 is ready to pair.
4. On the mobile computer, touch **Next**.

Figure 13 Waiting to Scan Screen



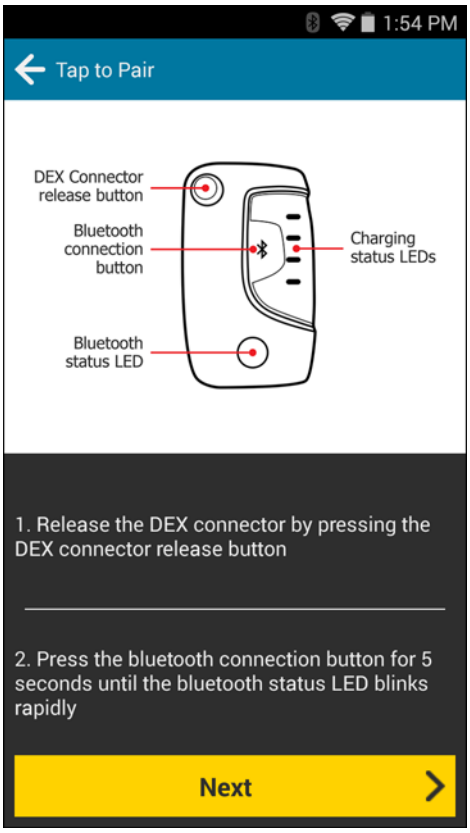
5. Scan the MAC Address bar code on the DX30. The devices begin to pair.
6. Upon successful pairing, the **Comm Settings** screen appears indicating the DX30 name and MAC address.
7. Touch the Back arrow button to return to the demo.

Tap to Pair

To tap to pair:

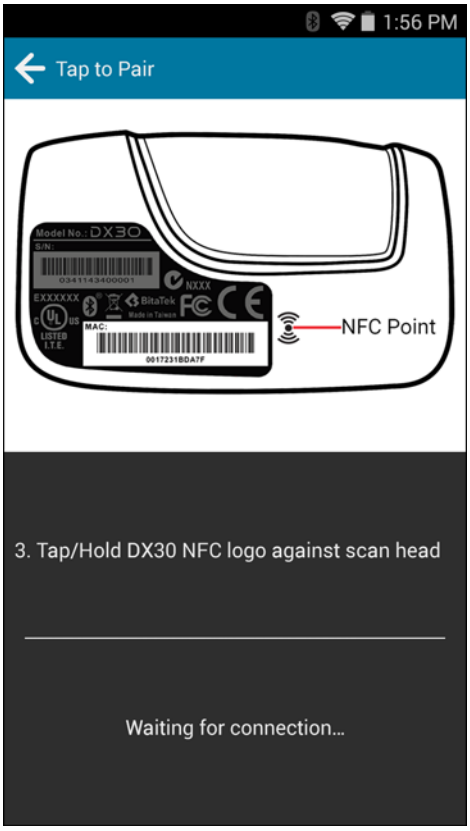
1. Touch **Tap to Pair.**
   The **Tap to Pair** screen appears instructing the user to prepare the DX30 for pairing.

Figure 14 Tap & Pair Screen



2. On the DX30, press the release button to open the DEX connector.
3. On the DX30, hold the pairing button down for five seconds. The blue light flashes for a second, stops flashing, and then continues to flash indicating the DX30 is ready to pair.
4. Touch **Next**.

Figure 15 Tap & Pair Confirmation Dialog Box



5. Tap the NFC symbol on the DX30 to the NFC symbol on the mobile computer. Note that not all Zebra mobile computers have an NFC icon on the housing. Refer to the mobile computer User Guide for information on using NFC.

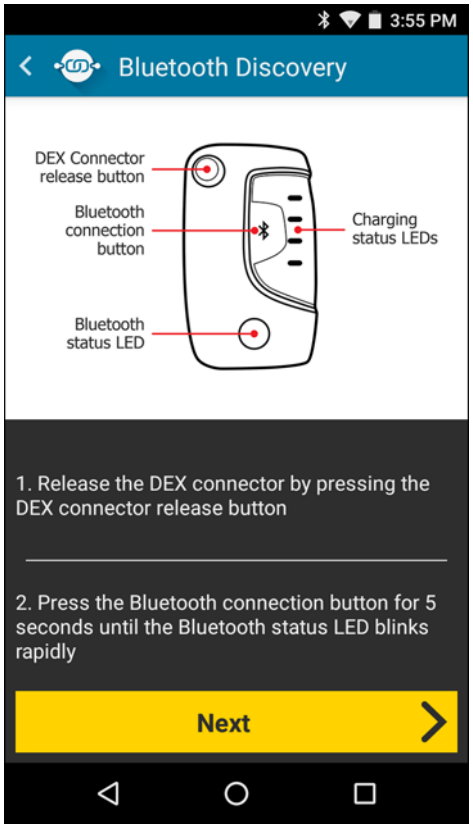
   When the user taps the devices together, the devices pair.
6. Upon successful pairing, the **Comm Settings** screen appears indicating the DX30 name and MAC address.
7. Touch the Back arrow button to return to the demo.

Discover to Pair

To discover and pair:

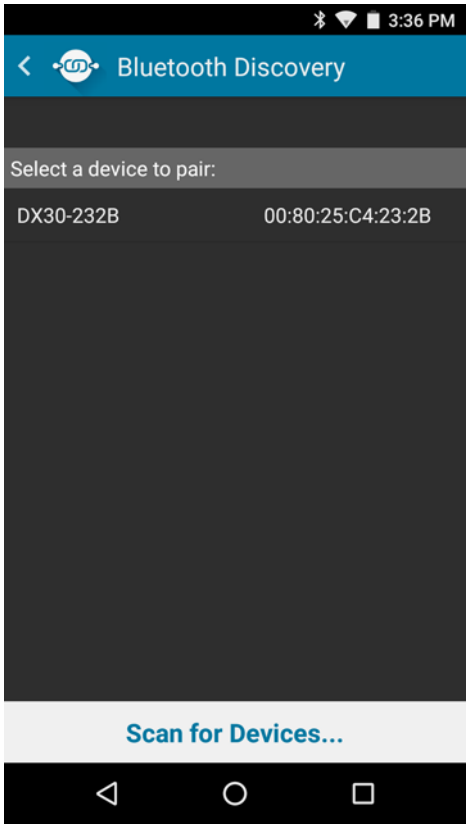
1. Touch **Discovery**.T he **Bluetooth Discovery** screen appears.

Figure 16 Bluetooth Discovery Screen



2. On the DX30, press the release button to open the DEX connector.
3. On the DX30, hold the pairing button down for five seconds. The blue light flashes for a second, stops flashing, and then continues to flash indicating the DX30 is ready to pair.
4. Touch **Next**. The devices searches for devices in the area and lists them on the screen.

Figure 17 Bluetooth Discovery Screen

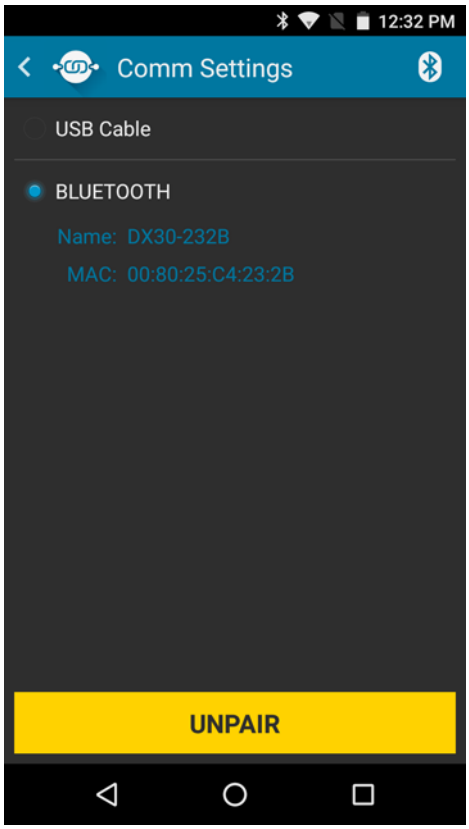


5. Touch the DX30 in the list to pair.
6. Upon successful pairing, the **Comm Settings** screen appears indicating the DX30 name and MAC address.
7. Touch the Back arrow button to return to the demo.

DX30 Bluetooth Key Fob Control Settings

At the **SmartDEX Comm Settings** screen, if the mobile computer is currently paired to a DX30, the radio button shows the DX30 is selected.

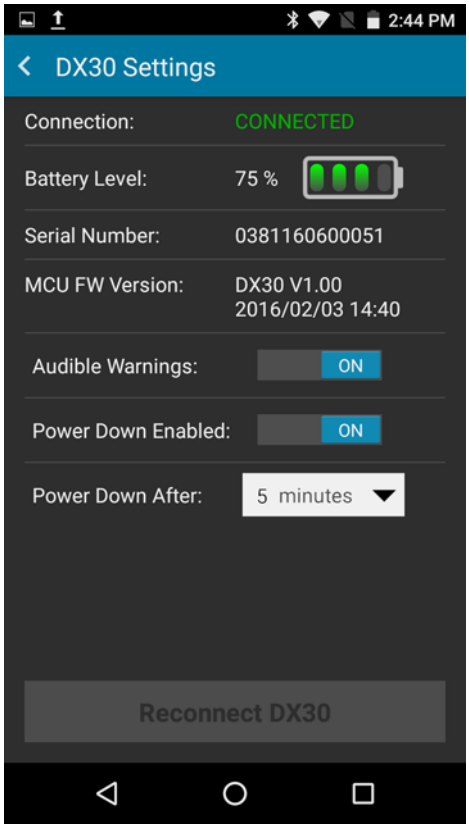
Figure 18 SmartDEX Comm Settings Screen



- A status message displays the DX30 ID and MAC address.

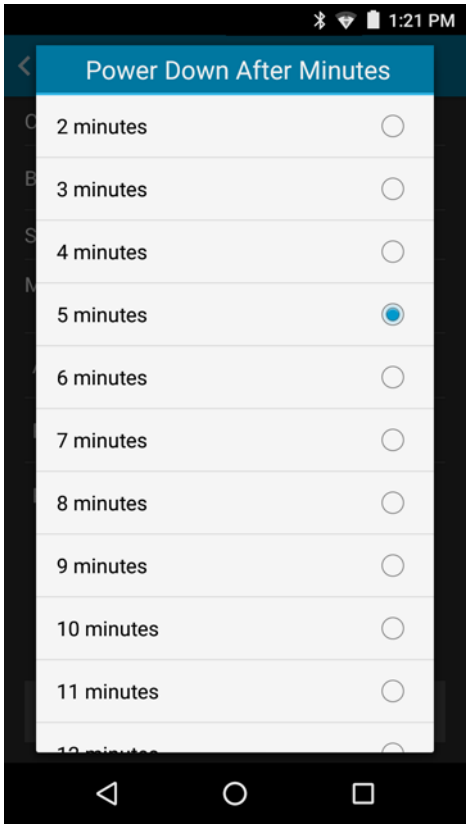
1. Tap the **Bluetooth** icon in the menu bar to display the user editable DX30 options.

Figure 19 SmartDEX Comm Settings Screen



- **Connection** - Displays the connection of the device and DX30.
- **Battery Level** - Indicates the DX30 battery level.
- **Serial Number** - Indicates the DX30 serial number.
- **MCU FW Firmware** - Indicates the DX30 firmware version.
- **Audible Warnings** - allows the user to turn on or turn off the DX30 beeping feature. Touch the switch to select **ON** or **OFF**. The DX30 beeps when the DX30 times-out and powers down or if the user leaves it behind (Default: On).
- **Power Down Enabled** - allows the user to select the number of minutes the DX30 waits after the last activity for the DX30 to power down.
- **Power down DX30 after** - Touch the text field. A menu appears.

Figure 20 Power Down Menu



2. Select the number of minutes the DX30 should wait from the last activity until powering down. Swipe up and down to see all options. The available range is 2 to 15 minutes in one minute increments (Default: 5 minutes).

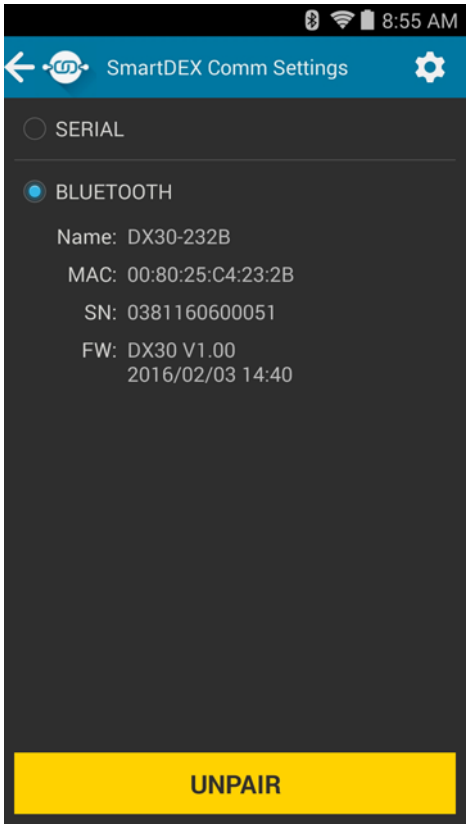
   Touch the back button, even if they have made a selection, no change will be saved, and the menu closes.

ELeash Settings

This ELeash feature is performed by the mobile computer. Use the ELeash settings to enable audio and vibration alerts when the mobile computer and DX30 are out of Bluetooth range. This is useful to notify the user if they left the DX30 behind. The DX30 has it's own electronic leash feature that alerts the user when the paired mobile computer is out of range. Both features work to ensure that the user does not leave the DX30 behind.

At the **SmartDEX Comm Settings** screen, if the mobile computer is currently paired to a DX30, the radio button shows the DX30 is selected.

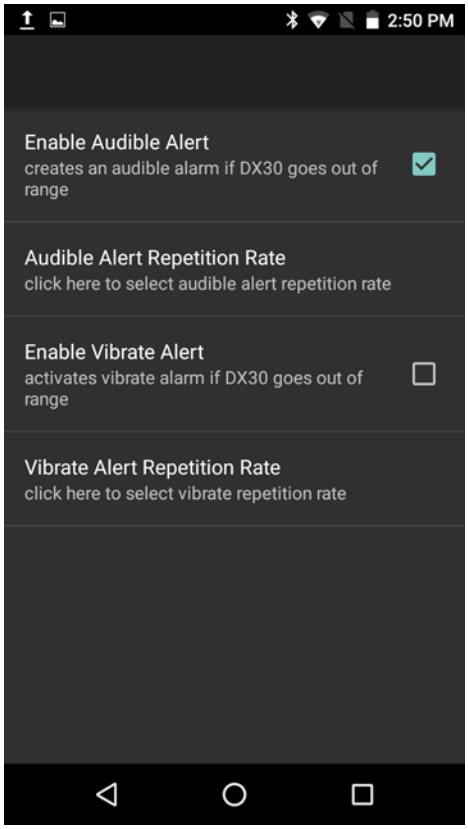
Figure 21 SmartDEX Comm Settings Screen



- A status message displays the DX30 ID and MAC address.

1. Touch the **Bluetooth** icon in the menu bar to display the user editable DX30 options.
2. Touch **ELeash** Settings. The ELeash settings screen appears.

Figure 22 ELeash Settings Screen



- **Enable Audible Alert** - Creates an audible alert when the DX30 goes out of Bluetooth range (default - enabled).
- **Audible Alert Repetition Rate** - Select the audible alert repetition rate (default - 3 seconds).
- **Enable Vibrate Alert** - Creates a vibration alert when the DX30 goes out of Bluetooth range (default - enabled).
- **Vibrate Alert Repetition Rate** - Select the vibration alert repetition rate (default - 3 seconds).

3. Touch Back button to return to the previous screen.

Error Reporting via Zebra Analytics

Tracing software problems encountered by field personnel is always a difficult task. To make this process easier SmartDEX on Android is able to generate an Error Report and transmit via Zebra Analytics engine to the Zebra Support team.

To send an error report:

1. When a error occurs, a message displays asking the user if they would like to report the error.
2. Touch **Yes** to send report or **No** to cancel.

Figure 23 SmartDEX Error Report Screen

Results Data

DEX Results data is retrieved from the intent passed into the DEX Receiver. See Implementation for details on how to obtain this data.

The format of the SmartDEX Results data consists of one or more lines containing information (segments) pertaining to invoice or line-item adjustments made by either the user (sender) and/or the DEX server (receiver).

Each segment identifier is separated by at least one (or more) space characters.

See Results Data Format more information.

DEX Client Data Format

Introduction

The route data consists of readable (printable) ASCII characters, with each line terminated with a NEWLINE character (x10). Empty lines and comments are ignored by the DEX parser. All others should be considered as command lines.

Command lines start with a **COMMAND** tag, followed by at least one space character, followed by one or more parameter arguments as required. Each parameter there after must be separated by at least one (1) SPACE character (x20). If a single parameter (or argument) contains one or more spaces, it must be enclosed inside double quotes.

Example: **DES "ABC Fun Corp. LLC"**

The route data content is divided into specific sections (please note the lines that **BEGIN** and **END** a section in the following format descriptions). The sequential ordering of each section must follow there rules:

There must be only one (1) VENDOR section, which will be followed by a single STOP section (standalone versions may contain multiple STOPs). Each STOP section will include one (1) CUSTOMER section, followed by one (or more) INVOICE sections. Each INVOICE section will include one (or more) ITEM sections. Each ITEM section may include one (or more) ADJUSTMENT commands.

The sequential ordering of commands within each section are described below(*);

```
VENDOR section
   (any order)
STOP section
   *must begin with STOP x
   CUSTOMER section
      (any order)
   INVOICE section
      *must begin with INVOICE x
      ITEM section
         *must begin with PRODUCT or SKU
         *next line must be UPC or GTIN
         (any order after that)
         *any ADJUSTMENT commands must be at the end

      more ITEM sections..
```

```
   more INVOICE sections..
more STOP sections (standalone only)
```

Command Tag Parameter and Qualifier Identifications

The following descriptors are used in the route file structure definition to help detail the content structure and format requirement for each of the route data command tags.

Table 2 Command Tag Descriptors

Item	Description
[]	brackets indicate a required command argument
()	parentheses indicate any optional command arguments

Argument qualifiers are followed by a numeric (max width) value.

Table 3 Command Tag Parameters and Qualifiers

Item	Description
A	alpha only - case sensitive Note 1: If content contains 1 or more spaces, enclodr within double quotes to avoid trunction. Note 2: Any '*' characters will be converted to '+' example A, ANTYTHING example; A7, SEVENTH example; A12, "THIS IS GOOD"
N	numeric [0-9] example; N, 0123456789012 example; N4, 1234
AN	alpha-numeric - combination of alpha and numeric (such as hexidecimal) example; AN, A1F9BB401 example: AN6, RT0012
D	decimal [0-9] [.] example; D3, 765.123 - will only accept 3 digits right of decimal.
B	boolean [TY1FN0] T|Y|1 = TRUE, F|N|0 = FALSE; Note: Only the first character is used to determine the boolean state of the argument. examples; True, T, False, F, Yes, Y, No, N, 11, 1, 00, 0
#	hashtags are used to indicate that any following text (on that line) is a comment. The hashtag must be the first character on that line. example; # THIS IS A COMMENT

All Command Tags referenced are MANDATORY, except where identified as being *OPTIONAL.

```
-- BEGIN VENDOR SECTION--
ROUTE [AN15]
Identifier for the route containing 1 or more stops.
e.g. ROUTE  "R1505"


NAME [AN30]
*OPTIONAL
Name of the Vendor/Sender
e.g. NAME "Area Dairies, Inc."


DUNS [N9]
Number assigned by Dun & Bradstreet (Data Universal Numbering System)
Identifies the vendor to the customer/receiver
e.g. DUNS  235887334


LOCATION [AN6]
Number that when assigned with sender DUNS provides uniqueness
e.g. LOCATION "SEA"


COMM_ID [N10]
ID assigned by UCC (uniform code council)
Identifies the transmitting company/vendor.
note: a company phone number is commonly used here
e.g. COMM_ID  9876543210


DEX VERSION [N4]
*OPTIONAL
Version of the EDI standard being used by the DEX server
Default = 4010 (earlier versions not supported)
Usage here will set the default value for all stops on this route
e.g. DEX_VERSION 4010

--END VENDOR SECTION--
```

```
--BEGIN STOP SECTION--

STOP [N3]
A sequential reference of one or more stop sections.
The first stop should be 1, and all subsequent values must increment by 1.
(embedded versions only use the first stop)
e.g. STOP 1

CUST_DUNS [N9]
The customers DUNS number
e.g. CUST_DUNS  232345018

CUST_LOCATION [N6]
Identifies the customer location being delivered to
e.g. CUST_LOCATION "2234"

CUST_NUMBER [N10]
*OPTIONAL
A reference number to identify the customer
e.g. CUST_NUMBER  2435670101

CUST_COMM_ID [N10]
*OPTIONAL
ID assigned by UCC (uniform code council)
e.g. CUST_COMM_ID  2435670101

CUST_NAME [AN30]
*OPTIONAL
Customer name
e.g. CUST_NAME  "WALMART SC 5577"

CUST_ADDR [AN30]
*OPTIONAL
Customer address
e.g. CUST_ADDR  "101 32ND ST"

CUST_CITY [AN20]
*OPTIONAL
Customer city
e.g. CUST_CITY  "Great Hills"

CUST_STATE [AN2]
```

*OPTIONAL
Customer state
e.g. CUST_STATE "WA"

CUST_ZIP [AN10]
*OPTIONAL
Customer zipcode
e.g. CUST_ZIP "98336"

CUST_PHONE1 [AN16]
*OPTIONAL
Customer phone
e.g. CUST_PHONE1 "(800) 123-4567"

CUST_PHONE2 [AN16]
*OPTIONAL
e.g. CUST_PHONE2 “(360) 435-5555”

CUST_CONTACT [AN30]
*OPTIONAL
e.g. CUST_CONTACT “Peter Parker Jr”

CUST_NOTES [AN50]
*OPTIONAL
e.g. CUST_NOTES “Deliver before 10am. Use side entrance in alley.”

--ADDITIONAL STOP COMMAND OPTIONS-
These OPTIONAL, stop specific commands override your DEX server default values. If the command line is missing, the default values are used.

DEX_VERSION [N4]
*OPTIONAL
Used here, will override the default DEX_VERSION (for this STOP only)
e.g. DEX_VERSION 5010

TEST_DATA [B]
*OPTIONAL
If set TRUE, flags the route data in this stop section is to be handled as TEST data. If set FALSE, the route data in this stop section is to be handled as live data.
e.g. TEST_DATA T

PROMPT_BEFORE_ACK [B]

*OPTIONAL

If set TRUE, prompts user before sending an automatic acknowledgment to the DEX receiver. If set FALSE, no prompt is given.

e.g. PROMPT_BEFORE_ACK YES

ACK_G88_ONLY [B]

*OPTIONAL

Send a SIGNATURE ONLY acknowledgment if a G88 only adjustment is received. If set FALSE, no SIGNATURE ONLY acknowledgment is sent for a G88 only adjustment.

e.g. ACK_G88_ONLY YES

ENABLE_G8602 [B]

*OPTIONAL

If set TRUE, 895 will include the G8602 segment. If set FALSE, will not include the G8602 segment.

e.g. ENABLE_G8602 TRUE

ENABLE_895_G84 [B]

*OPTIONAL

If set TRUE, 895 will include the G84 record. If set FALSE, will not include the G84 record.

e.g. ENABLE_895_G84 Y

ENABLE_895_G8403 [B]

*OPTIONAL

If set TRUE, 895 will include the G8403 segment. If set FALSE, will not include the G8403 segment.

e.g. ENABLE_895_G8403 Y

ENABLE_895_G84_ADJONLY [B]

*OPTIONAL

If set TRUE, 895 will include a G84 record - only when adjustments exist. If set FALSE, will not include a G84 record.

e.g. ENABLE_895_G84_ADJONLY Y

EXCLUDE_895_G84_HCODE_06 [B]

*OPTIONAL

If set TRUE, excludes the G8406 segment, if G84 exists. If set FALSE, includes the G8406 segment, if G84 exists.

e.g. EXCLUDE_895_G84_HCODE_06   1

REJECT_ADJ_SVR_NEW_ITEM [B]

*OPTIONAL

If set TRUE, rejects server NEW ITEM adjustments. If set FALSE, accepts server NEW ITEM adjustments.

e.g. REJECT_ADJ_SVR_NEW_ITEM YES

REJECT_ADJ_ITEM_COST_REDUCTION [B]

*OPTIONAL

If set TRUE, rejects ITEM adjustments that reduce vendor cost. If set FALSE, accepts ITEM adjustments that reduce vendor cost.

e.g. REJECT_ ADJ_ITEM_COST_REDUCTION YES

PRESERVE_UPC12 [B]

*OPTIONAL

If set TRUE, ensures all UPC12 values are correct format. If set FALSE, does not ensure all UPC12 values are correct format.

e.g. PRESERVE_UPC12 YES

--END STOP SECTION--

--BEGIN INVOICE SECTION--

INVOICE [AN22]

invoice identifier - MUST BE UNIQUE

e.g. INVOICE "961226099001"

ORDER_TYPE [A8]

DEX base record type - DELIVERY or RETURN (default = DELIVERY)

e.g. ORDER_TYPE DELIVERY

ORDER_NUMBER [AN22] [N8]

*OPTIONAL

Customer purchase order and date

ORDER_NUMBER "34637113433" 20150326

USER_CAN_ADJUST_QTY [B]

*OPTIONAL

If set TRUE, allows the user to edit/adjust item quantities. If FALSE, the user is not allowed to edit/adjust item quantities.

e.g. USER_CAN_ADJUST_QTY TRUE

```
USER_CAN_ADJUST_COST [B]
*OPTIONAL
**default can be set by licensing server, but use here will override
If set TRUE, allows the user to edit/adjust item cost.
e.g. USER_CAN_ADJUST_COST TRUE

USER_CAN_ADJUST_UOM [B]
*OPTIONAL
**default can be set by licensing server, but use here will override
If set TRUE, allows the user to edit/adjust item unit of measure.
(can only adjust while status is UNSENT)
e.g. USER_CAN_ADJUST_UOM TRUE

INVADJUSTMENT [A1] [N3] [N2] [N16] [A1] [D4] (N4) (A2)
*OPTIONAL
Invoice adjustment (max 20 per Invoice-parsing errors result if more than 20)
Argument 1          adjustment type: A=allowance, C=charge
Argument 2          adjustment code (UCS Code 340)
Argument 3          handling code (UCS Code 331)
                    01=Bill Back, 02=Off Invoice, 15=Info Only
Argument 4          vendor code (your reference id)
Argument 5          value flag: T or %
                    T to specify a total price allowance/charge (requires a cost value)
                    % to specify a percentage allowance/charge
                    (requires 2 additional values: cost/percentage)

e.g. INVADJUSTMENT C 525 02 246809753 T 23.45
e.g. INVADJUSTMENT A 235 02 987654321 % 35.00/10.5

--END INVOICE SECTION--
--BEGIN ITEM SECTION--

PRODUCT [AN15]  or SKU [AN15]
Vendor reference to an item - must be first command.
e.g. PRODUCT  "PR23425"
e.g. SKU "123456"

=====================================================
SUNRISE 2005 GUIDLINE FOR 5010UCS VERSION
Starting with DEX VERSION 5010, GTIN data is preferred.
The industry favors the use of GTIN data instead of UPC.
The GTIN command is not supported for DEX_VERSION <  5010
=====================================================
```

GTIN [N14] (N14) (N4) (N3)

Or

UPC [N12] (N12) (N4) (N3)

Product detail - must follow PRODUCT or SKU

The syntax for both commands is the same - however, UPC cannot support 14 digit GTIN values under UCS4010. If using GTIN data, be sure to use the GTIN command, and make sure the DEX version is set to 5010 or greater.

When used to reference a CASE, the first parameter (inner item) is always required. It is acceptable to insert the same value in both the 1st and 2nd parameters if there is no separate value for the inner items.

Argument1: barcode value for single item

Argument2: barcode value for case item.
If this param exists, PACKTYPE will default to CA(case)

Argument3: case pack count - number of inner containers or eaches

Argument4: case inner pack count - number of eaches per inner container
GTIN value includes prefix and check digit
GTIN type automatically determined by length of data
EO = 8, UP = 12, EA = 13, UK = 14

e.g. GTIN 06671355001234 (an each)

e.g. GTIN 06671355001234 06671356001234 12 4 (a case containing 12 containers of 4 items)

e.g. UPC 112233445577 (an each)

e.g. UPC 500020447568 540020447777 12 6 (a case containing 12 containers of 6 items)

DESC [AN20]

Text description of the item/product.

e.g. DESC "CHERRY VANILLA ICE CR"

QTY [N4]

Number of items or cases ordered (or returned)

e.g. QTY 10

PRICE [D2]

Price for each unit (packtype) in dollars and cents.

PRICE 5.95

PACKTYPE [A2]

Pack type - 'unit of measure' of item being delivered (or returned)

Default = EA (each) - UCS UOM Code 355

PACKTYPE EA

The most common pack types are: EA(each), CA(case), LB(pounds), DZ(dozen)
GA(gallon), KE(keg).

ADJUSTMENT [A1] [N3] [N2] [N16] [A1] [D4] (N4) (A2)

*OPTIONAL

Item adjustment (maximum of 10 per Item-parsing errors result if more than 10)

Argument 1	adjustment type: A=allowance, C=charge
Argument 2	adjustment code (UCS Code 340)
Argument 3	handling code (UCS Code 331) 01=Bill Back, 02=Off Invoice, 15=Info Only
Argument 4	vendor code (your reference id)
Argument 5	flag: T or % or $ T to specify a total price allowance/charge (requires a cost value) % to specify a percentage allowance/charge (requires 2 additional values: cost/percentage) $ to specify a rate per quantity (requires 2 additional values: cost/quantity)

e.g. ADJUSTMENT C 525 02 246809753 T 23.45

e.g. ADJUSTMENT A 235 02 987654321 % 35.00/10.5

e.g. ADJUSTMENT A 88   02 123456789 $ 2.00/5

PRESERVE_COST [B]

*OPTIONAL

If set TRUE, rejects server adjustments that reduce the vendor cost. If False, or missing, accepts server adjustments that reduce the vendor cost.

e.g. PRESERVE_COST YES

--END ITEM SECTION--

Additional INVOICE sections (and accompanying ITEM SECTIONS) go here…

Additional STOP sections (with accompanying INVOICE & ITEM SECTIONS) go here…

Results Data Format

Introduction

In EMBEDDED mode, DEX Results data is retrieved from the intent passed into the DEX Receiver. See Implementation for details on how to obtain this data.

The format of the SmartDEX Results data consists of one or more lines containing information (segments) pertaining to invoice or line-item adjustments made by either the user (sender) and/or the DEX server (receiver).

Each segment identifier is separated by at least one (or more) space characters.

Example:

```
140701:015830 894:USR 1007 ADJ_QTY    2   1   2
140701:015830 894:USR 1007 ADJ_QTY    4   1   4
140701:015830 895:SVR 1007 ADJ_LOCATION 102 100
140701:015830 895:SVR 1007 ADJ_ALLOWANCE  1 -0.3500
140701:015830 895:SVR 1007 ADJ_QTY    2   2   1
140701:015830 895:SVR 1007 INVC_STATUS    3
```

File Format for Results Data

[TMSTAMP] [UCS_TYPE:CODE] [INVOICE ID] [ADJ_TYPE] [PARAMS]

[TMSTAMP] [UCS_TYPE:CODE] [INVOICE ID] [ADJ_TYPE] [PARAMS]

[TMSTAMP] [UCS_TYPE:CODE] [INVOICE ID] [ADJ_TYPE] [PARAMS]

…

Where:

TMSTAMP	datetime value - YYMMDD:hhmmss
UCS_TYPE	message affected by adjustment - 894 or 895
*CODE	reason code. If **codes.dat** does not exist, defaults to USR or SVR
INVOICE	invoice number
ADJ_TYPE	(ADJ_NEW_ITEM, ADJ_QTY, ADJ_PRICE,…)

PARAMS as necessary

Example:

```
140701:015830 894:013 1007 ADJ_QTY    2   1   2
140701:015830 894:014 1007 ADJ_QTY    4   1   4
140701:015830 895:SVR 1007 ADJ_LOCATION 102 100
140701:015830 895:SVR 1007 ADJ_LOCATION 102 100
140701:015832 895:USR 1007 INVC_STATUS 3
```

Adjustment Descriptions

Item Level Adjustments

✓ **NOTE:** lineitem always refers to the 894 base record sequential number of the line item (G8201) for the identified invoice (G8202).

ADJ_PACKTYPE	lineitem	old_value	new_value
ADJ_PRICE	lineitem	old_value	new_value
ADJ_QTY	lineitem	old_value	new_value
ADJ_UPC	lineitem	old_value	new_value
ADJ_CASEUPC	lineitem	old_value	new_value
ADJ_PACK	lineitem	old_value	new_value
ADJ_INNERPACK	lineitem	old_value	new_value
ADJ_PROD_QUALIFIER	lineitem	old_value	new_value
ADJ_PROD_ID	lineitem	old_value	new_value

ADJ_ALLOWANCE	lineitem	rate
ADJ_CHARGE	lineitem	rate
ADJ_KILL_PREVIOUS_ALLOW_CHG	lineitem	

All previous allowances/charges are removed for this item.

ADJ_NEW_ITEM

A new item has been added to the invoice (lineitem = last lineitem + 1)

(4010) lineitem upc qty packtype price pack inner_pack case_upc

(5010) lineitem prod_qualifier prod_id qty packtype price pack inner_pack case_qualifier case_id

(5030) lineitem prod_qualifier prod_id qty packtype price pack inner_pack case_qualifiercase_id

ADJ_NEW_ITEM_REJECTED

Client setting was set to reject any server created new items

Params:

(4010) prod_id qty unit_of_measure item_list_cost

(5010) prod_id_qualifier prod_id qty unit_of_measure item_list_cost

NOTE: in versions greater than 4010 the UPC / CASE_UPC fields are deprecated, and replaced by the two new fields – prod_id_qualifier and prod_id.

ADJ_DEL_ITEM lineitem

Item was removed from invoice

NOTE: lineitem values refer to the initial sequential ordinal value of each item belonging to an invoice defined by the route data. The sequential order of a lineitem does not change during the DEX transactions, thereby maintaining referential alignment. The ADJ_DEL_ITEM notify's the recipient that the retailer has REMOVED this lineitem from the invoice.

INVC_STATUS_MANUALLY_CHANGED old_value new_value
User manually changed invoice status.

Invoice status values:

0 = UNSENT
1 = SENT
2 = RECEIVED
3 = CLOSED

Invoice Level Adjustments

NOTE: Invoice level adjustments do not contain a lineitem reference.

ADJ_INVC_ALLOWANCE rate

ADJ_INVC_CHARGE rate

ADJ_INVC_KILL_PREVIOUS_ALLOW_CHG

All previous invoice allowances/charges are removed.

Server Only Generated Adjustments

These adjustments are considered informational only and do not affect any lineitem values.

ADJ_D_R_DATE old_ value new_ value

Actual or intended date of physical delivery or return expressed in format CCYYMMDD.

ADJ_POTD old_value new_value

Date which is meaningful for both supplier and distributor for various mutually defined purposes (i.e., date of product ownership transfer) expressed in format CCYYMMDD.

ADJ_PONUM old_value new_value

Identifying number for Purchase Order assigned by the orderer/purchaser.

ADJ_PODATE old_value new_value

Date assigned by the purchaser to Purchase Order expressed in format CCYYMMDD.

ADJ_LOCATION old_value new_value

Number assigned by the customer that when combined with receiver Duns number uniquely identifies the receiving location.

SmartDEX Licensing

Licensing

Under the SmartDEX licensing model, Zebra sells a group of licenses for a specific period of time, in annual increments. The following terms will be used to describe SmartDEX licensing:

- **Customer account** - the user's main account. A customer account must have at least one sub-account, but may often have more than one sub-account.
- **Sub-accounts** - Logical or regional sub-groupings of license seats.
- **License Period** - the period of time in which the licenses are active.
- **Activation year** - the initial license period. This year always costs more than extension years.
- **Extension years** - additional license periods sold in addition to the Activation year.
- **Activation date** - the date upon which license seats in a sub-account are activated.
- **Expiration date** - the date upon which license seats in a sub-account expire and become inactive.
- **Device registration** - the point at which a device is registered with the licensing server, and is ready to receive a license seat.
- **License entitlement** - the individual license leased to an individual mobile device.
- **Entitlement lease** - the period of time that a license entitlement is assigned to a mobile device. Typically between 10 and 14 days.
- **Entitlement lease expiration date** - the date upon which a entitlement lease expires.
- **Entitlement range** - the total number of license entitlements across all Sub-accounts for a single Customer Account.

Customer Accounts

SmartDEX is licensed to the customer account as a number of License Entitlements in one or more sub-account. The customer account is a logical container for the sub-accounts.

Sub Accounts

License Enablements are grouped into Sub-Accounts. Sub-Accounts may have different Activation dates, upon agreement with Zebra and Versatile. This allows the client to roll-out devices to geographical or logical company divisions on an extended schedule. At least one Sub-Account must be activated within 90 days of receipt of the SmartDEX hardware.

Each Sub-Account holds a specific number of License Entitlements - the License Entitlements pool. All licenses within Sub-Account have the same Activation date and expiration date. This is true no matter how many License

Entitlements are actually leased to devices in the field. For instance, if a customer has a Sub Account with fifty License Entitlements, but never activates more than 45, the last five seats expire with all the other seats upon the expiration date.

Individual handheld devices are activated through a Sub-Account. The Versatile Licensing Server (VLS) tracks all activated devices.

License Periods

SmartDEX licenses are sold in one year increments. Each set of license entitlements is sold with an initial Activation year and optional Extension years, which renew and extend the license period for some number of years after the Activation Year. For example the purchase of an Activation year and two Extension years would give the customer three years of service. While the license period is in effect, the customer receives full software support and software updates as they are released.

Activation Dates

The activation date for the software licenses is normally the date the customer takes possession of the SmartDEX hardware. However, upon agreement with Zebra and Versatile, the activation date may be delayed up to 90 days if necessary.

Expiration Dates

When the expiration date is reached without renewal SmartDEX becomes disabled until a new Activation year is purchased. The customer may purchase Extension years anytime before the expiration date to receive the discounted Extension year rate. Sub-accounts that do not purchase Extension years prior to the Expiration date must purchase a new Activation year at the higher rate.

Device Registration

Each device must obtain a valid license activation from the Versatile Licensing Server (VLS). Activation confirms that the device has SmartDEX properly installed and set up. It also notifies VLS that the device is eligible to receive a license entitlement lease if one is available. Every DEX Client must obtain a valid activation to unlock its functionality.

There are two (2) modes of activation available:

- Activate Now - consumes a license entitlement (if available) and unlocks the client
- Stage Only - does not consume a license, but allows pre-staging of the client (requires a subsequent license refresh before DEX operations are accessible).

License Entitlement Leases

We use the concept of the License Entitlement Lease as a way to ensure devices communicate with the licensing server on a regular basis. As an individual device is activated, it leases one of the License Entitlements. When all of the License Entitlements have been disbursed to specific devices, no new device activations will be allowed until a License Entitlement leases expires, and becomes available. Lease periods may be set to any length of time, but should be at least 10 days long, and no longer than 14 days.

Each time SmartDEX opens, it checks to see if it has an active seat entitlement. If it does not, SmartDEX operations are disabled until the lease is renewed. The user sees a warning screen, and is instructed to connect to the internet and renew their seat lease.

This process is important, as it ensures all devices communicate at least once during the lease period, allowing the server to update the software, and track all active devices.

When all of the licenses in a Sub-Account have been activated, no more handhelds may be activated through that Sub-Account without purchasing more licenses. If a handheld requires repairs, it should be deactivated to free up a license seat lease for a replacement device. If a handheld becomes lost, stolen or is damaged beyond repair, the device's License Seat will become available upon expiration its current seat lease. If the device serial number is known, the seat lease can be manually revoked, and the license seat activated on a different device without waiting for the seat lease to expire.

Implementation

Implementation Guideline

The embedded version of SmartDEX for Android is designed to be launched by a separate task or activity running Android 4.1.1 or later. To fully support this model, it is necessary to create a **Launcher** (for starting the client), and to register a **BroadcastReceiver** for listening (to detect when the client has finished).

The BroadcastReceiver intent is the object used to obtain returned information from the DEX client. The following descriptions detail the supported data that can be accessed from the intent.

```
int mode = intent.getIntExtra("mode");
   Returns the mode that initiated the call (0 or 1)
   0 = DEX CLIENT STARTED
   1 = DEX CLIENT REFRESH


int status = intent.getIntExtra("status");
   Returns the status of the operation.
   0 = OPERATION SUCCESSFUL
   20-30 = PARSE ERRORS
   70-71 = WIRELESS ERRORS
   80-89 = DEX COMM ERRORS
   9xx = EXCEPTION ERROR


int lic_status = intent.getIntExtra("lic_status");
   Returns the status of the device license.
   0 = ACTIVE
   41 = NOLICENSE
   42 = INVALID SN
   43 = BAD FORMAT
   44 = INVALID DATE RANGE
   45 = EXPIRED
   46 = DISABLED
   47 = LIC 3 DAY SUSPENSION
```

```
String sResults = intent.getStringExtra("results");
   Returns the results data
String sActivity = intent.getStringExtra("activity");
   Returns the activity log of the initiated session
```

There are many ways to integrate SmartDEX for Android - the code samples below demonstrates one such way to apply this into the Android application.

Refreshing the Client License Transparently

The embedded version of SmartDEX also supports an interface for allowing the peer program to initiate a DEX client license refresh in the background. Refreshing a client license ensures that the device is ready and able to actively DEX when needed. The life-cycle of a client license stays active for a period of 15 days from the date of the last refresh or registration, and may be refreshed as often as desired. Most typically, refreshes are performed either daily or weekly, at start of day, on devices that intend to DEX that day or week. The 15 day cycle allows usage for times when no Wifi or cellular connectivity is available.

The implementation guidelines contain the additional detail for initiating the client refresh. It is the peer programs responsibility (or the user's) to ensure that either Wi-Fi or cellular data is enabled before initiating the call. If internet access to the licensing server is not available, the license cannot be refreshed.

There are many ways to integrate SmartDEX for Android - the code samples that follow demonstrate one such way to apply this into your Android application. Refer to the sample method: **dex_finished()** for details on the values available via the returned intent.

Launcher

```
// Example (android) code for basic startup of SmartDEX

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;



//=============================================================================
public class MyActivity extends Activity{

   private final String PACKAGE_NAME = "com.vms.android.VersatileDEX";
   private final int ACTION_START_DEX = 0;
   private final int ACTION_START_REFRESH = 1;
   private DexClientListener mDexClientListener = null; //from DEX client listener

   // handler for receiving the intent from the DEX client listener
```

```
private final Handler hDexFinished = new Handler() {
   @Override
   public void handleMessage(Message msg) {
      switch (msg.what) {
         case DexClientListener.DEX_FINISHED:
            {
               final Intent intent = (Intent) msg.obj;
               if (null ! = intent) {
                  handleDexFinished(intent);
               }
            }

            break;

         default:
            super.handleMessage(msg);
      }
   }
};

//-----------------------------------------------------------------------------
@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.YOUR_LAYOUT_RESOURCE); // <-- put your layout resource here

   mDexClientListener = new DexClientListener(hDexFinished);
   IntentFilter filter = new IntentFilter(DexClientListener.ACTION_DEX_FINISHED);
   registerReceiver(mDexClientListener, filter);
}//onCreate

//-----------------------------------------------------------------------------
private void StartDEX(){
   String sRouteData = getRouteDataAsString(); // your method call to get String route
   content
   Intent intent = new Intent( Intent.ACTION_SEND  );
   intent.setType( "text/dex_route_data" );
   intent.setPackage(PACKAGE_NAME);
   intent.putExtra( "mode", ACTION_START_DEX); //start dex mode
   intent.putExtra( Intent.EXTRA_TEXT, sRouteData ); //

   startActivity(intent);
```

```
    }

    //---------------------------------------------------------------------------
    private void handleDexFinished(Intent intent){
        // the default value of -999 is used here to indicate an UNDEFINED reference

        // read values from returned intent
        int mode = intent.getIntExtra("mode", -999);
        int status = intent.getIntExtra("status",-999);
        int lic_status = intent.getIntExtra("lic_status", -999);
        int comm_status = intent.getIntExtra("comm_status", -999);
        String sResults = intent.getStringExtra("results");
        String sActivity = intent.getStringExtra("activity");

        // process the values...
    }

    //---------------------------------------------------------------------------
    private String getRouteDataAsString() {
        // build the route data here..
        String sData = "build the route data here..";
        return sData;
    }

    //---------------------------------------------------------------------------
    @Override
    public void onDestroy(){
    if(null != mDexClientListener){
        unregisterReceiver(mDexClientListener);
    }
}


// this class is used to send the intent to whatever listener is listening
//=============================================================================
public class DexClientListener extends BroadcastReceiver {
    // handler used to send the received intent
    private final Handler handler;

    // this value can be whatever you want it to be - this is just an example
    public static final int DEX_FINISHED = 0x777;
```

```
    // this string must not be changed
    public static final String ACTION_DEX_FINISHED =
    "com.vms.android.versatiledex.ACTION_DEX_FINISHED";

    //------------------------------------------------
    public DexClientListener(Handler handler){
    this.handler = handler;
    }

    //------------------------------------------------
    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();

        if (action.equals(ACTION_DEX_FINISHED)) {
            handler.obtainMessage(DEX_FINISHED, intent).sendToTarget();
        }
    }
}
```