Near Field Communications (NFC) Manager: BL652 Sample *smart*BASIC application

Application Note *v1.0*

Introduction

The goal of this document is to introduce the Near Field Communication (NFC) functionality in the BL652 module and show how to utilise the functionality with the **NFC Manager** *smart*BASIC application by testing the functionality over the UART and introduces ideas for implementing NFC in designs.

Potential Use-Cases for NFC on BL652

- Embedded advertising: NFC can be used to interact with customers by having tags placed at bus stops or convention areas which can open websites when tapped for customer engagement. By leveraging the BL652, instead of having a read only disposable tag that needs replacing each time the data requires updating, the Bluetooth Low Energy functionality of the BL652 can be used to update the tag data and only allow connections from trusted devices. This ensures that tags cannot be erased or modified by unauthorised users or devices.
- Automatically launching applications when a tag is scanned: Windows Mobile and Android both support NFC tags which launch your application or show the application in the marketplace if it is not installed on the device already. This helps a user save time installing or launching an application.
- Waking up via NFC field sense: The BL652 NFC interface can be configured so that the module can be put into deep sleep mode and be awakened by using an NFC reader on the tag. This can be used to keep the module in a very low power mode prolonging battery life for embedded applications or mixed with other deep sleep wakeup methods to only have the BL652 executing in response to input from users.

Overview

NFC Manager provides a command interface over the UART for interfacing with the NFC functionality of the BL652 module. It allows you to enable/disable NFC and add/change/delete NDEF (NFC Data Exchange Format) messages which can be read by other devices. The BL652 NFC functionality currently allows for read-only (type 2) tags. It can be used to send specific and custom data to an NFC reader such as mobile device application launchers, website URL openers and messages that can show on the screen of the device. Note that each mobile operating system and device is configured differently, so different devices may not utilise NFC in the same way.

In this sample application, the BL652 will be used to create NFC tags which can be configured and activated over the UART, and tested with a mobile Android device that has NFC support (some Windows Phone/Mobile devices have NFC support and should work similarly to the Android devices. Apple iOS devices with NFC chipsets are not currently supported as the NFC functionality is only available for wireless payments and cannot be used by applications).

Note: Laird provides a library of sample *smart*BASIC applications including **NFC Manager**, to provide a simple, easy-to-use guide for implementing a range of different functionality within your applications. The sample application library on GitHub is never intended to be a completely robust, end-customer application for use in real world applications.

Note: The BL652 has support for one NFC connection acting as a read-only type 2 tag supporting a data rate of 106 kbps.

Note: For further information on NFC and the BL652, please read the NFC section in the BL652 *smart*BASIC manual which has a very detailed description on how NFC works and how it is implemented in the BL652 module.

REQUIREMENTS

- DVK-BL652 development kit
- PC with a spare USB port (using a USB Hub if appropriate)
- Android/Windows phone device with NFC functionality (or an alternative NFC reading device, like an Arduino with an NFC Shield)
- UwTerminalX – available for Windows, Linux and Mac: https://github.com/LairdCP/UwTerminalX/releases
- **NFC Manager** sample app – https://github.com/LairdCP/BL652-Applications/blob/master/Applications/nfc.manager.sb

Note: For the purposes of this document, we assume you have familiarised yourself with compiling/loading *smart*BASIC applications.

NFC OVERVIEW

NFC (Near Field Communication) is a short range wireless communication method suitable for low powered devices that transmit a relatively small amount of data. It uses a 13.56 MHz carrier to transmit data with a range of up to 20 cm but most NFC devices have a much smaller range of <5 cm. For further details on NFC, please see the BL652 *smart*BASIC extension manual.

Like other *smart*BASIC functionality, NFC can be controlled on the BL652 in a state-machine based system with 3 distinct segments (Figure 1):

1. Setting up NFC:
    - The NFC hardware is initialised, the NDEF messages are created and committed and then the NFC field is enabled. Whenever a device reads the BL652 NFC tag, an optional *smart*BASIC event will be thrown which indicated that the NFC field was established, broken or that reading the tag has finished.
2. Changing NFC data
    - The data on the NFC tag is currently read-only to remote devices but the data can be changed from the BL652 by disabling the NFC field, deleting the NDEF messages and creating the new messages then committing them to the stack and finally enabling the NFC field.
3. Closing NFC
    - The NFC tag can be closed by disabling the field, deleting the messages and closing the NFC handle. The NFC device can be reinitialised at any time. It is important to note the difference between the *reset* and *delete* NFC functions: to free the memory used by NFC back for use by

other functionality it needs to be *deleted*. *Resetting* the memory only clears the data so that it does not have to be reallocated for it to be used by an NFC message again.

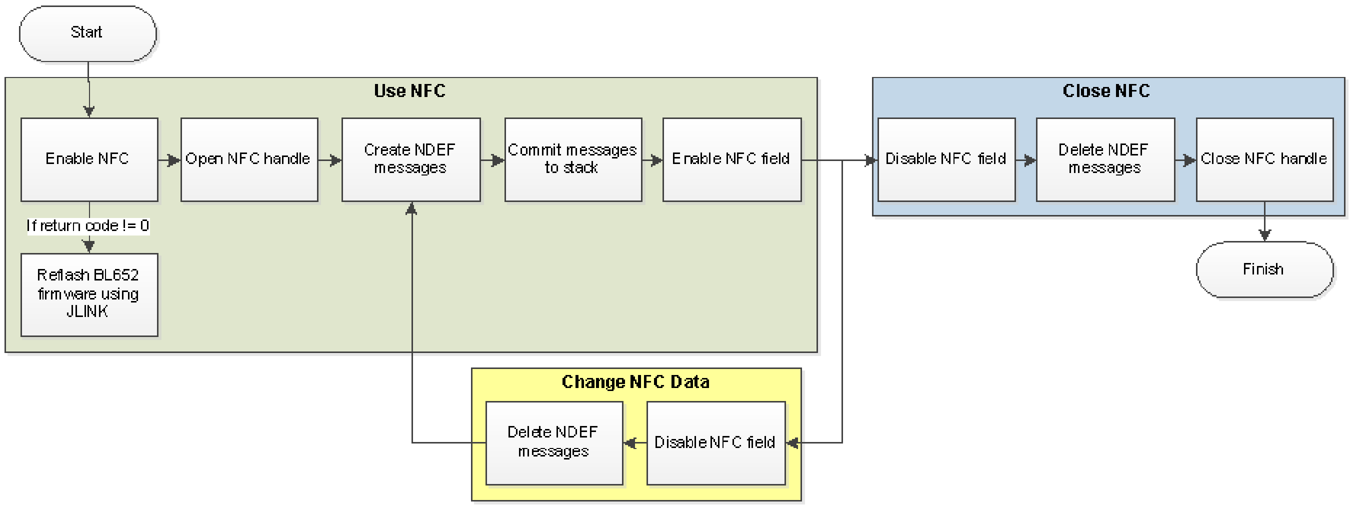
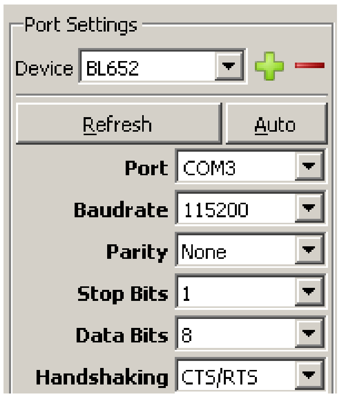


Figure 1: NFC in smartBASIC overview

BL652 Development Kit Setup

To set up the development kit, follow these steps:

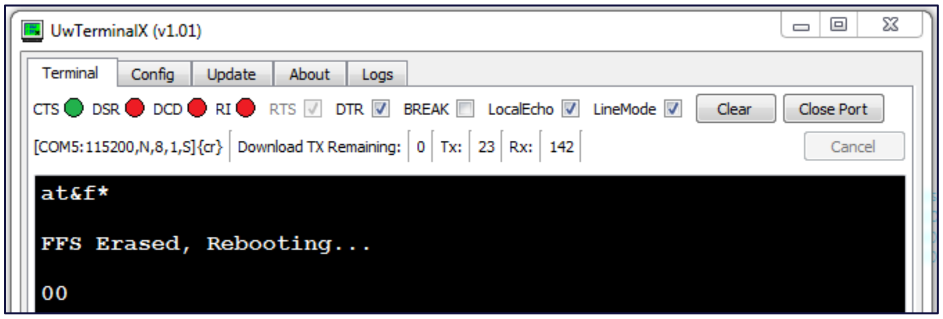
1. Connect your BL652 Development kit to your PC via the USB Micro cable. The power LED illuminates when the board is receiving power.
2. Open UwTerminalX. In the Config tab, set the parameters and COM port associated with your development board.



Device	BL652
Port	COM3
Baudrate	115200
Parity	None
Stop Bits	1
Data Bits	8
Handshaking	CTS/RTS

Figure 2: Setting connection parameters

3. Click **OK** to advance to the Terminal tab.
4. Use UwTerminalX to return the BL652 to factory defaults using the command *at&f** as shown (Figure 3). If you are using a new development board with the sample application, you may need to remove the autorun jumper on J12 and press the reset button to exit out of the sample application, and then issue the *at&f** command to erase the file system and all non-volatile data.



```
at&f*

FFS Erased, Rebooting...

00
```

Figure 3: Factory default

5. Load **NFC Manager** – use the right-click menu to select **XCompile + load**.

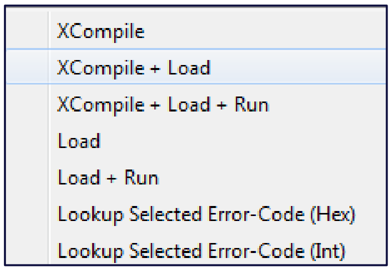

XCompile
XCompile + Load
XCompile + Load + Run
Load
Load + Run
Lookup Selected Error-Code (Hex)
Lookup Selected Error-Code (Int)

Figure 4: XCompile Load and Run

6. Select the **nfc.manager.sb** file which was downloaded and extracted from the Zip file from Github or saved from the Github raw file page: https://raw.githubusercontent.com/LairdCP/BL652-Applications/master/Applications/nfc.manager.sb.
7. Wait for the NFC manager program to load; this should take approximately 10 seconds.

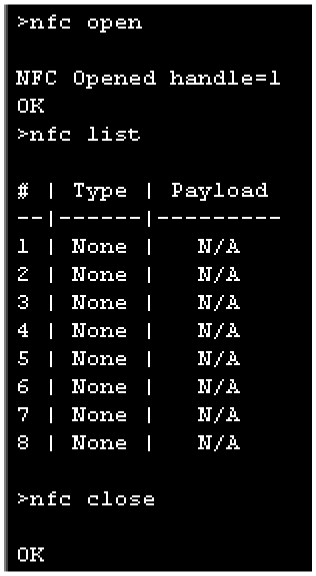
NFC Manager can now be run by typing ***nfc*** followed by return.

Note: A complete list of commands available through NFC manager can be found at the top of the source code file, documented later in the source code file by searching for **#CMD#**, or below in the Full List of NFC Manager Commands section.

OPENING NFC

After launching the application, the application is in a neutral state whereby the NFC device is not open. To perform any NFC operations, the NFC sub-system needs to be opened which returns a handle on which to perform operations. Do this by sending the command '**nfc open**' over the UART. As the BL652 only has a single NFC device, the handle returned will always be the same value. Once open, the NFC device can be closed using '**nfc close**'.

By default, there are no NDEF messages when the NFC device is opened, the list of NDEF messages can be outputted by sending the command '**nfc list**' **(**Figure 5**)**



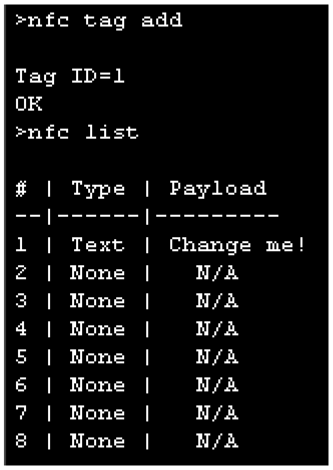
```
>nfc open

NFC Opened handle=1
OK
>nfc list

# | Type | Payload
--|------|---------
1 | None |   N/A
2 | None |   N/A
3 | None |   N/A
4 | None |   N/A
5 | None |   N/A
6 | None |   N/A
7 | None |   N/A
8 | None |   N/A

>nfc close

OK
```

Figure 5: Basic NFC functions

Command	Description	smartBASIC Command
nfc open	Opens the NFC sub-system hardware	NFCOpen()
nfc close	Closes the NFC sub-system hardware	NFCClose()
nfc list	Lists the NDEF message types and data payloads	

CREATING NDEF MESSAGES

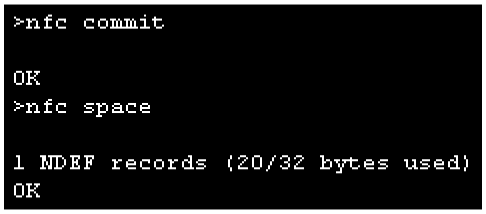
The first step to creating NDEF messages is to ensure that the NFC device is open using '**nfc open**', then a new tag can be added using '**nfc tag add**' (up to 8 NDEF messages can be created using this application). Once a new message has been added, the tag ID is returned and the new tag can be viewed by using the command '**nfc list**' (Figure 6).



```
>nfc tag add

Tag ID=1
OK
>nfc list

# | Type | Payload
--|------|---------
1 | Text | Change me!
2 | None |   N/A
3 | None |   N/A
4 | None |   N/A
5 | None |   N/A
6 | None |   N/A
7 | None |   N/A
8 | None |   N/A
```

Figure 6: Adding a NDEF Message

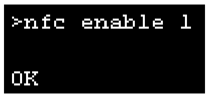
At this stage the NDEF message has not been committed, commit the data to the stack using '**nfc commit**'. With the data committed, information about the NDEF records can be retrieving using '**nfc space**' which will show the total space used and allocated for the NFC tags in bytes (Figure 7).



```
>nfc commit

OK
>nfc space

1 NDEF records (20/32 bytes used)
OK
```

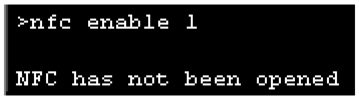
Figure 7: Commiting the NFC tag and vieiwing the space used

The final step before reading the data is to enable the NFC field sense – once enabled the data in the tag becomes read only. It cannot be changed from *smart*BASIC until NFC is disabled, when the data can then be changed and the NFC field enabled again. The NFC field is enabled using '**nfc enable 1**' and disabled using '**nfc enable 0**' (Figure 8).



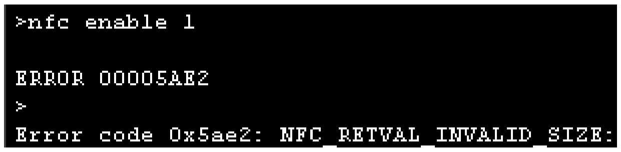
```
>nfc enable 1

OK
```

Figure 8: NFC field enabled



```
>nfc enable 1

NFC has not been opened
```

Figure 9: Error if NFC isn't open



```
>nfc enable 1

ERROR 00005AE2
>
Error code 0x5ae2: NFC_RETVAL_INVALID_SIZE:
```

Figure 10: Error if no NDEF tags have been added

Command	Description	smartBASIC Command
nfc tag add	Adds an NDEF tag and returns the ID	
nfc tag remove #id#	Removes NDEF tag with supplied ID	
nfc tag payload #id# #data#	Updates the payload of NDEF tag of supplied ID to the supplied data	
nfc list	Lists the NDEF message types and data payloads	
nfc space	Shows the total free/used space of the committed NDEF tags	NFCNdefMsgGetInfo()

nfc commit	Commits the current list of NDEF messages to the NFC stack	NFCFieldSense()
		NFCNdefMsgNew()
		NFCNdefRecAddGeneric()
		NFCNdefMsgCommit()
		NFCNDefMsgDelete()

nfc enable 0	Disables the NFC tag (this is required before NDEF messages can be committed)	NFCFieldSense()
nfc enable 1	Enables the NFC tag and allows remote devices to read the data.	NFCFieldSense()

READING NFC TAGS ON ANDROID

Note: For this application note, a Samsung Galaxy Note 2 is used so the exact steps for utilising NFC functionality for your device may differ. Please check with your device manual for information on how to configure and use NFC.

On the phone, enable NFC functionality from the settings (Figure 11**Error! Reference source not found.**). Hold your mobile device flush against the BL652 NFC antenna (the range of NFC is very small - Figure 12) and the NFC tag will be read by the device.

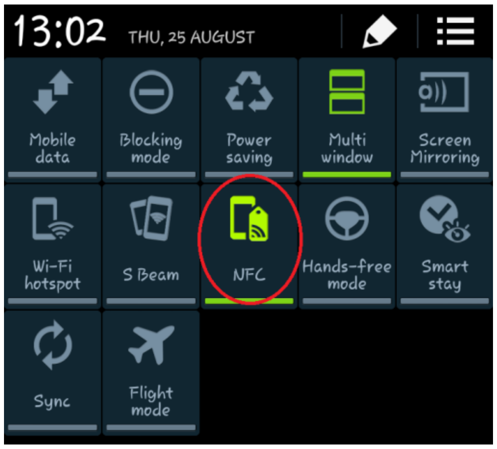


Figure 11: Enabling NFC on Android

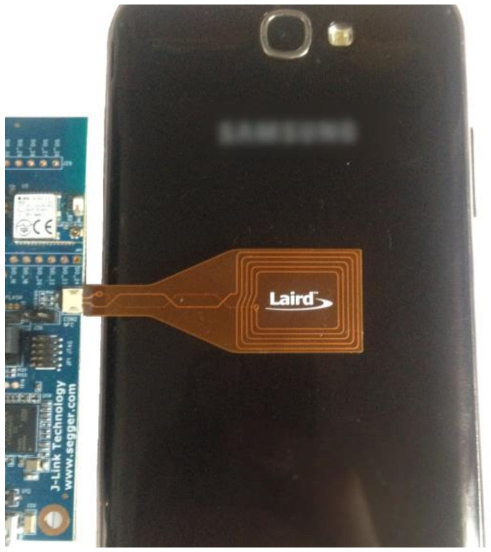
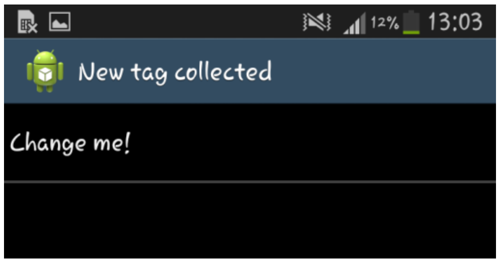


Figure 12: BL652 NFC antenna held flush against android device

The tag currently contains a simple text message which should be displayed on the screen (Figure 13).



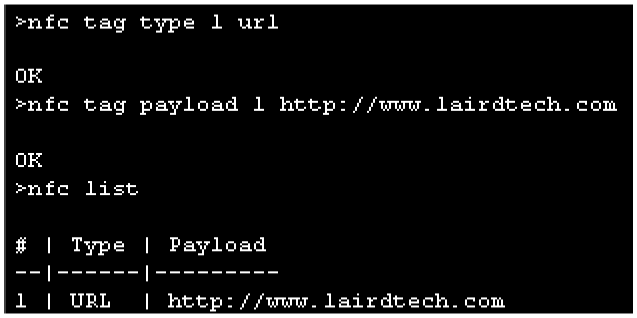
New tag collected

Change me!

Figure 13: Default text tag as detected on Android

LAUNCHING URLS IN A WEB BROWSER

The tag type can be changed to advertise a URL instead of just text. Some devices will automatically open a web browser on the page indicated by the tag, other devices will ask the user for a confirmation before visiting the site. The tag type can be changed to a URL by sending the command '**nfc tag type 1 url**' where 1 represents the tag ID. As only one tag has been added, 1 is currently the only valid tag. The URL also needs to be set using '**nfc tag payload 1 http://www.lairdtech.com**' (Figure 14**Error! Reference source not found.**) and the updated NDEF message can be updated by issuing the command '**nfc commit**'. Now if you hold your phone up to the tag you should be greeted with an option to open the URL or be taken to the URL automatically (Figure 15**Error! Reference source not found.**)**.**



```
>nfc tag type 1 url

OK
>nfc tag payload 1 http://www.lairdtech.com

OK
>nfc list

# | Type | Payload
--|------|---------
1 | URL  | http://www.lairdtech.com
```

Figure 14: Changing the NDEF message to a URL and setting the URL

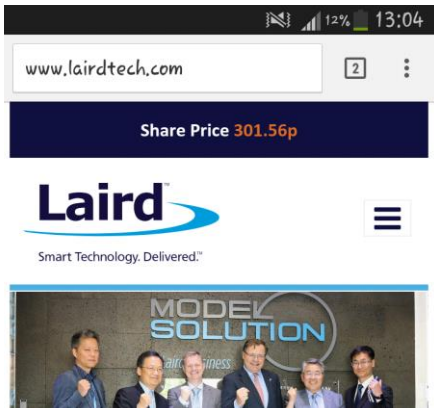
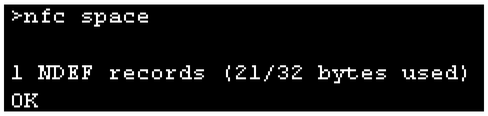


Figure 15: Website opened via URL NDEF Message

What is interesting to check at this point is the space used by the NDEF messages in the stack by issuing the command '**nfc space**' (Figure 16) – notice that the size is only 21 bytes but the length of the URL itself is 24 bytes long – how can the entire NDEF message with a URL be shorter than the URL itself?



```
>nfc space

1 NDEF records (21/32 bytes used)
OK
```

Figure 16: NDEF URL message size

The reason for this is that the NDEF URI (Uniform Resource Identifier) format has a special format that allows URIs to take up much less space by having the first byte of the NDEF URI message indicate the start protocol and address – saving space for embedded devices and requiring less memory. Inside the **NFC Manager** application source code, the process of shorterning the URI is done in the *CompressURL$* function. Some of the most common protocols are listed in the following table:

First byte of NDEF URI	URI prefix
0x00	*No prefix*
0x01	http://www.
0x02	https://www.
0x03	http://
0x04	https://
0x06	mailto:
0x0D	ftp://
0x18	btspp://
0x19	btl2cap://
0x1A	btgoep://
0x1B	tcpobex://
0x1C	irdaobex://
0x1D	file://

As a result: http://www.lairdtech.com is shortened into \01lairdtech.com – saving 10 bytes! This URI shortening system is part of the NDEF specifications and is universally supported.

Command	Description	smartBASIC Command
nfc tag type #id# URL	Changes NDEF tag of supplied ID to be a URL	
nfc tag payload #id# #data#	Updates the payload of NDEF tag of supplied ID to the supplied data	

nfc commit	Commits the current list of NDEF messages to the NFC stack	NFCFieldSense()
		NFCNdefMsgNew()
		NFCNdefRecAddGeneric()
		NFCNdefMsgCommit()
		NFCNDefMsgDelete()

Launching Android Applications

An NFC tag can be used to automatically launch an application on an Android device if the application is installed, or open the Google Play page for the application if it is not installed. To do this you need the full application ID. For example, the Laird Toolkit application ID is *com.lairdtech.lairdtoolkit*. The original NDEF message can be changed to an Android application launcher by sending '**nfc tag type 1 AApp**' and the application to open can be set using '**nfc tag payload 1 com.lairdtech.lairdtoolkit**'. Once committed using '**nfc commit**' and tapping an Android device onto the NFC antenna, the Laird Toolkit application will open if installed (Figure 17) or the Google Play store will open showing the application page.

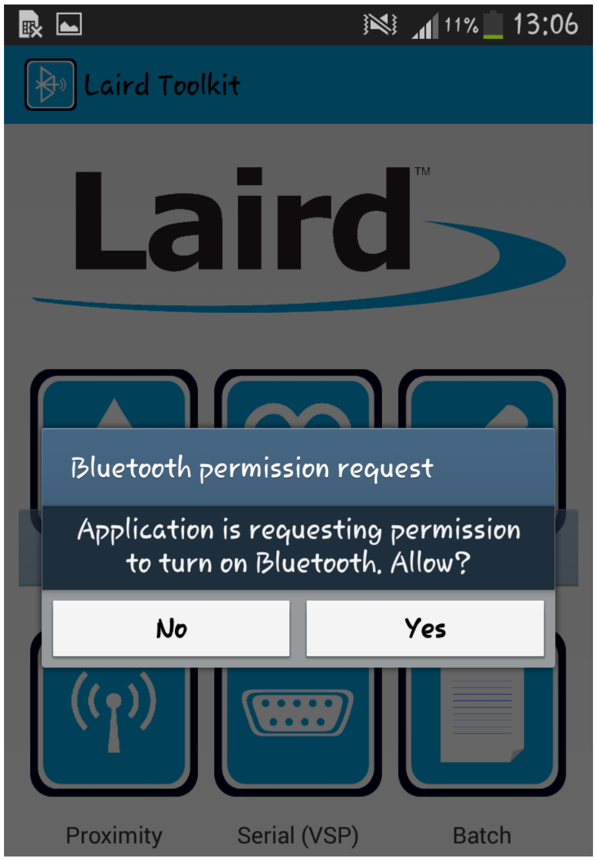

13:06
Laird Toolkit
Laird
Bluetooth permission request
Application is requesting permission to turn on Bluetooth. Allow?
No
Yes
Proximity
Serial (VSP)
Batch

Figure 17: Laird Toolkit launched via Android App Launcher NDEF Message

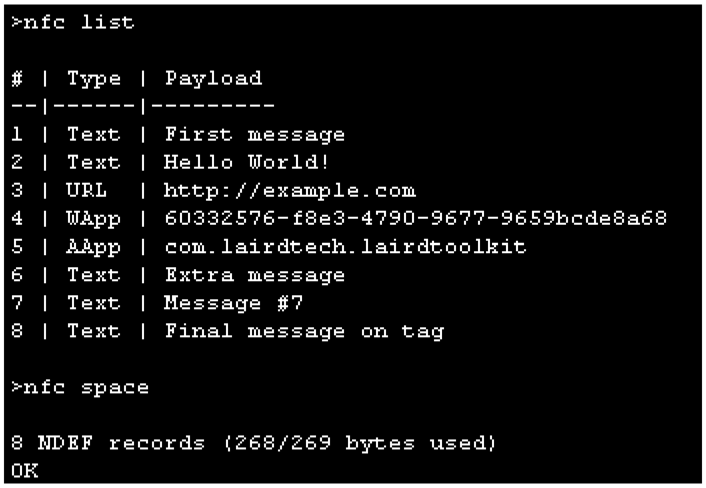
Command	Description	smartBASIC Command
nfc tag type #id# AApp	Changes NDEF tag of supplied ID to be an Android Application Launcher	
nfc tag payload #id# #data#	Updates the payload of NDEF tag of supplied ID to the supplied data	

nfc commit	Commits the current list of NDEF messages to the NFC stack	NFCFieldSense()
		NFCNdefMsgNew()
		NFCNdefRecAddGeneric()
		NFCNdefMsgCommit()
		NFCNDefMsgDelete()

Multiple NDEF Messages

Multiple NDEF messages can exist in a single tag, this for example could show text and open a website or provide application specific data and open an Android application which can read the data – there are many possibilities. Using the **NFC Manager** to create multiple messages is as simple as issuing the command '**nfc tag add**' which will add a new message and return the handle for the tag. Up to 8 messages can be created with the NFC manager application, but this is not a hardware limit and can be changed using a define in the application source code if needed (see the '*#define NUM_OF_NDEF*' line).

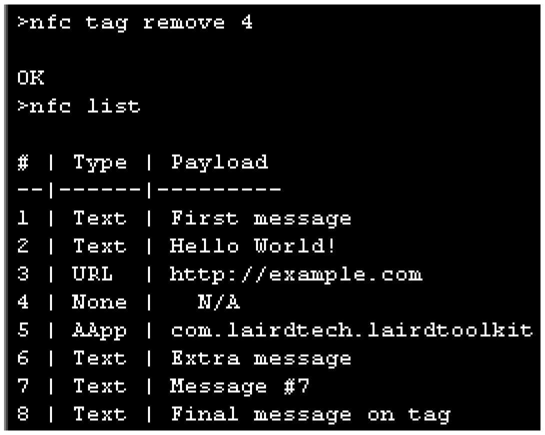
As explained previously, the messages are all independent and can have different message types and payloads using '**nfc tag payload #id #data**' and '**nfc tag type #id #type**' where #id is a number between 1-8, #type is: *Text*, *URL*, *AApp* or *WApp*, and #data is the payload for the message. A full NFC tag with 8 messages is shown in Figure 18



```
>nfc list

# | Type | Payload
--|------|----------
1 | Text | First message
2 | Text | Hello World!
3 | URL  | http://example.com
4 | WApp | 60332576-f8e3-4790-9677-9659bcde8a68
5 | AApp | com.lairdtech.lairdtoolkit
6 | Text | Extra message
7 | Text | Message #7
8 | Text | Final message on tag

>nfc space

8 NDEF records (268/269 bytes used)
OK
```

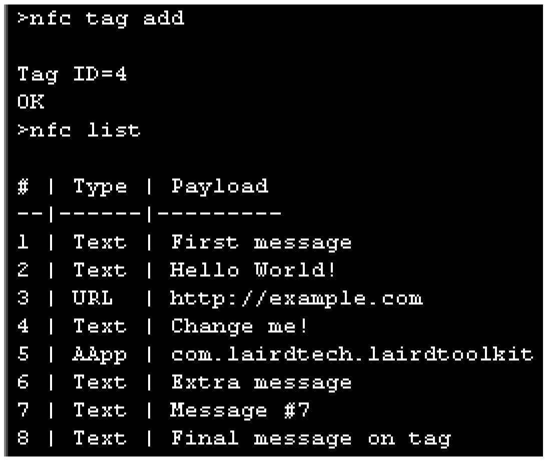
Figure 18: 8 message NFC tag

Messages can be deleted if they are no longer needed using '**nfc tag remove #id**' (Figure 19). The order of NFC messages will not change when a message has been deleted and if required the tag can be added back again using '**nfc tag add**' with the default type and value (Figure 20). Remember that once the messages have been changed, these changes will not appear on the tag until the '**nfc commit**' command is issued.



```
>nfc tag remove 4

OK
>nfc list

# | Type | Payload
--|------|---------
1 | Text | First message
2 | Text | Hello World!
3 | URL  | http://example.com
4 | None |   N/A
5 | AApp | com.lairdtech.lairdtoolkit
6 | Text | Extra message
7 | Text | Message #7
8 | Text | Final message on tag
```

Figure 19: Removing a tag



```
>nfc tag add

Tag ID=4
OK
>nfc list

# | Type | Payload
--|------|---------
1 | Text | First message
2 | Text | Hello World!
3 | URL  | http://example.com
4 | Text | Change me!
5 | AApp | com.lairdtech.lairdtoolkit
6 | Text | Extra message
7 | Text | Message #7
8 | Text | Final message on tag
```

Figure 20: Adding a tag back after deleing

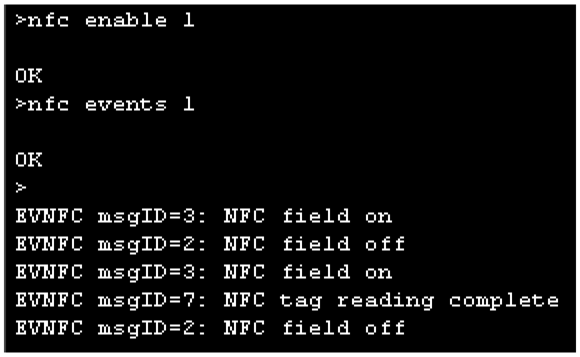
Command	Description	smartBASIC Command
nfc tag add	Adds an NDEF tag and returns the ID	
nfc tag remove #id#	Removes NDEF tag with supplied ID	
nfc tag type #id# text	Changes NDEF tag of supplied ID to be a text tag	
nfc tag type #id# URL	Changes NDEF tag of supplied ID to be a URL	
nfc tag type #id# WApp	Changes NDEF tag of supplied ID to be a Windows Application Launcher	
nfc tag type #id# AApp	Changes NDEF tag of supplied ID to be an Android Application Launcher	
nfc tag payload #id# #data#	Updates the payload of NDEF tag of supplied ID to the supplied data	
nfc list	Lists the NDEF message types and data payloads	
nfc space	Shows the total free/used space of the committed NDEF tags	NFCNdefMsgGetInfo()

nfc commit	Commits the current list of NDEF messages to the NFC stack	NFCFieldSense()
		NFCNdefMsgNew()
		NFCNdefRecAddGeneric()
		NFCNdefMsgCommit()
		NFCNDefMsgDelete()

nfc clear	Clears all NDEF messages and values (does not commit change to stack)	

NFC Status Messages

By default, NFC-related status messages are not printed out to the UART (these messages indicated when the NFC field is activated/deactivated or when the tag has been read). NFC status messages can be enabled by sending the command '**nfc events 1**' and disabled by sending '**nfc events 0**' (Figure 21).



```
>nfc enable 1

OK
>nfc events 1

OK
>
EVNFC msgID=3: NFC field on
EVNFC msgID=2: NFC field off
EVNFC msgID=3: NFC field on
EVNFC msgID=7: NFC tag reading complete
EVNFC msgID=2: NFC field off
```

Figure 21: NFC event messages when enabled

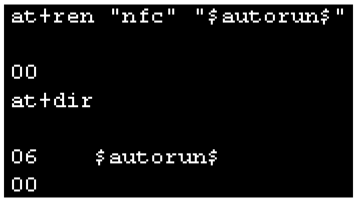
Command	Description	smartBASIC Command
nfc events 0	Disables NFC message outputting	OnEvent EVNFC disable
nfc events 1	Enables NFC message outputting	OnEvent EVNFC call HandlerNfc

Using NFC to Awaken BL652 in Deep Sleep Mode

NFC can be used to awaken a BL652 that is in Deep Sleep mode (a very low power mode). For this to work, NFC needs to be setup and enabled, with the application renamed as an autorun application.

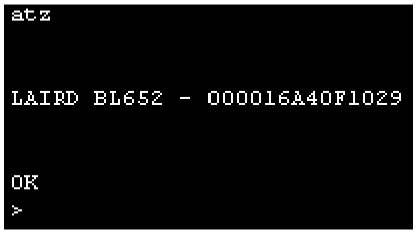
Begin by resetting the BL652, either by enabling and disabling BREAK from UwTerminal/UwTerminalX or pressing the reset button on the development board. Then rename the application so that it automatically runs on start-up by issuing the command '**at+ren "nfc" "$autorun$"**'. Confirm that the rename was successful by sending '**at+dir**' (Figure 22).

Ensure autorun mode is enabled by having the jumper on J12 of the development board on pins 1-2 and having DTR checked in UwTerminal/UwTerminalX. Then send the command '**atz**' and the **NFC Manager** application should start (Figure 23).



```
at+ren "nfc" "$autorun$"

00
at+dir

06    $autorun$
00
```

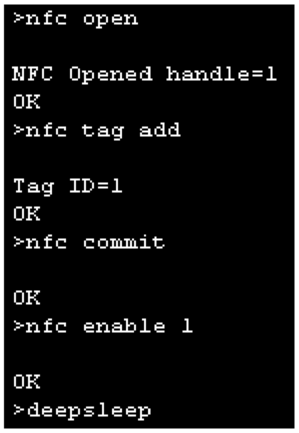
Figure 22: Renaming NFC Manager as an autorun application



```
atz


LAIRD BL652 - 000016A40F1029


OK
>
```

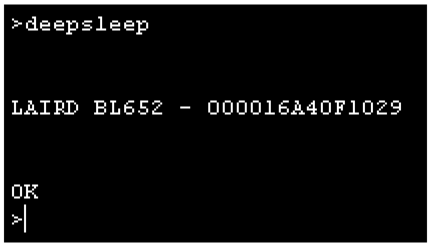
Figure 23: NFC Manager as an autorun application

When in deep sleep mode, the NFC data cannot be read by remote devices so only a simple message needs to be created. Open NFC with '**nfc open**', add a new sample tag using '**nfc tag add**' and commit to the stack using '**nfc commit**'. Then enable the NFC field with '**nfc enable 1**' and put the BL652 module into deep sleep mode by sending '**deepsleep**' (Figure 24) – the device will now be in its lowest power state and will not respond to any UART commands. To restart the device, hold an NFC reader over the NFC antenna. The module will reset and execute the autorun application, which is the **NFC manager** (Figure 25).



```
>nfc open

NFC Opened handle=1
OK
>nfc tag add

Tag ID=1
OK
>nfc commit

OK
>nfc enable 1

OK
>deepsleep
```

Figure 24: Setting up BL652 for NFC deep sleep waking



```
>deepsleep


LAIRD BL652 - 000016A40F1029


OK
>|
```

Figure 25: BL652 woken up from NFC field

To exit out from the autorun application, send ‘**exit**’ or ‘**quit**’ to return to command mode. The autorun application can be removed by deleting the file using ‘**at+del “$autorun$”**’ or by clearing the module and all non-volatile data by using ‘**at&f***’

Command	Description	smartBASIC Command
at+ren “nfc” “$autorun$”	(From command mode) Renames the application ‘nfc’ to ‘$autorun$’ so that it will be executed automatically at start-up when the autorun functionality is enabled	
atz	(From command mode) Resets the module	
nfc open	Opens the NFC sub-system hardware	NFCOpen()
nfc enable 1	Enables the NFC tag and allows remote devices to read the data.	NFCFieldSense ()
nfc tag add	Adds an NDEF tag and returns the ID	

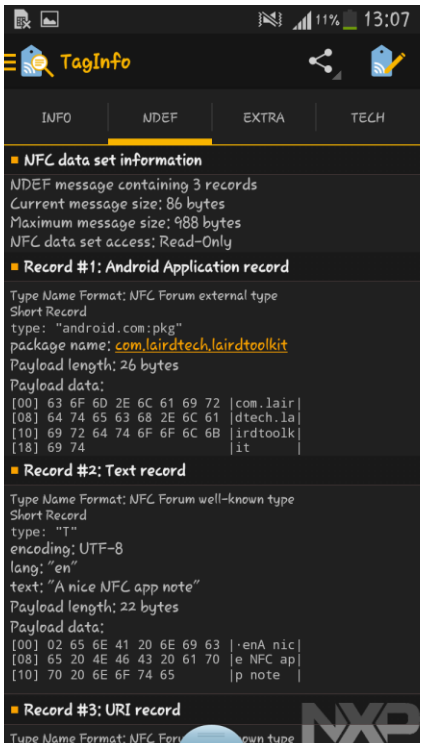
nfc commit	Commits the current list of NDEF messages to the NFC stack	NFCFieldSense()
		NFCNdefMsgNew()
		NFCNdefRecAddGeneric()
		NFCNdefMsgCommit()
		NFCNDefMsgDelete()

deepsleep	Puts module into deep-sleep mode (if NFC has been enabled and field sense is on then the device will awaken when an NFC field is detected).	SystemStateSet()
at+del “$autorun$”	(From command mode) Deletes the application named ‘$autorun$’ from the module. Note that the space used by deleted files is not freed up, the at&f* command is required to free the space up for use by other applications or files.	
at&f*	(From command mode) Clears the module and all non-volatile returning it factory settings.	
quit	Exit the application	
exit	Exit the application	

ADDITIONAL NFC TAG DETAILS/DEBUGGING

It may be necessary to debug NFC tags or view the raw data being sent. A useful application for this is NFC TagInfo by NXP Semiconductor, available on the Google Play app store: https://play.google.com/store/apps/details?id=com.nxp.taginfolite

When an NFC tag is scanned, all the individual NDEF records can be viewed showing the decoded data and a hex view of the raw data (Figure 26).



NFC data set information

```
NDEF message containing 3 records
Current message size: 86 bytes
Maximum message size: 988 bytes
NFC data set access: Read-Only
```

Record #1: Android Application record

```
Type Name Format: NFC Forum external type
Short Record
type: "android.com:pkg"
package name: com.lairdtech.lairdtoolkit
Payload length: 26 bytes
Payload data:
[00] 63 6F 6D 2E 6C 61 69 72 |com.lair|
[08] 64 74 65 63 68 2E 6C 61 |dtech.la|
[10] 69 72 64 74 6F 6F 6C 6B |irdtoolk|
[18] 69 74                   |it      |
```

Record #2: Text record

```
Type Name Format: NFC Forum well-known type
Short Record
type: "T"
encoding: UTF-8
lang: "en"
text: "A nice NFC app note"
Payload length: 22 bytes
Payload data:
[00] 02 65 6E 41 20 6E 69 63 |·enA nic|
[08] 65 20 4E 46 43 20 61 70 |e NFC ap|
[10] 70 20 6E 6F 74 65       |p note  |
```

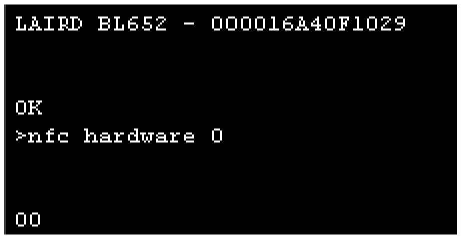
Record #3: URI record

```
Type Name Format: NFC For        own type
```

Figure 26: NXP NFC TagInfo, an application which provides detailed information on NFC tags

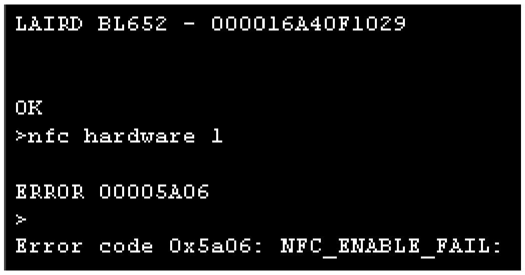
DISABLING NFC FUNCTIONALITY IN BL652

When NFC functionality is enabled, which is by default, two IO pins on the BL652 module are dedicated for use as the NFC antenna (SIO_9 and SIO_10) with active field input protection. NFC functionality can be disabled by a *smart*BASIC application to remove the NFC protection from the pins, allowing them to be used as normal GPIO pins. To do this, send '**nfc hardware 0**' with the **NFC Manager** application (Figure 27). Please note that once the NFC pin functionality has been disabled, it can only be re-enabled by flashing the BL652 firmware using the JLINK interface (UART upgrade will not work) – see the BL652 *smart*BASIC extension manual for further details. If NFC functionality has been disabled, this application will not work and an error will be returned when an attempt is made to use the NFC *smart*BASIC functions (Figure 28).



```
LAIRD BL652 - 000016A40F1029


OK
>nfc hardware 0


00
```

Figure 27: Disabling NFC functionality



```
LAIRD BL652 - 000016A40F1029


OK
>nfc hardware 1

ERROR 00005A06
>
Error code 0x5a06: NFC_ENABLE_FAIL:
```

Figure 28: Error returned when NFC is disabled

Command	Description	smartBASIC Command
nfc	Runs the loaded **NFC Manager** program	
nfc hardware 0	Disables the NFC hardware and resets the module	NFCHardwareState()
nfc hardware 1	Will return OK if the NFC hardware is enabled or an error if it is disabled	NFCHardwareState()

How the Application Works

The fundamental NFC manager application works based upon the initial state diagram shown in the overview section (Figure 1). When the NDEF messages are committed, the size of each message is calculated (including the shortening of the URLs) and a new NFC tag is created that is this size (or set to 32 bytes if the total data is smaller than this as 32 bytes is the minimal size). Each record is then added and committed to the stack (in a normal application it is not required to have an exact utilisation, for example 128 bytes may be allocated and at most 80 bytes may be used by NDEF messages – this is perfectly fine). Because NDEF messages cannot be changed if the NFC field is active, a secondary tag variable is used, once all the data has been committed then this tag is made active and the previous tag (if there was one) is deleted.

Full List of NFC Manager Commands

The following lists contains all the NFC commands available in the **NFC Manager** application:

Command	Description	smartBASIC Command
nfc open	Opens the NFC sub-system hardware	NFCOpen()
nfc close	Closes the NFC sub-system hardware	NFCClose()
nfc hardware 0	Will disable the NFC hardware and reset the module (requires JLINK flash to re-enable)	NFCHardwareState()
nfc hardware 1	Will return an error if the NFC hardware is disabled	NFCHardwareState()
nfc enable 0	Disables the NFC tag (this is required before NDEF messages can be committed)	NFCFieldSense()
nfc enable 1	Enables the NFC tag and allows remote devices to read the data.	NFCFieldSense ()

Command	Description	smartBASIC Command
nfc tag add	Adds an NDEF tag and returns the ID	
nfc tag remove #id#	Removes NDEF tag with supplied ID	
nfc tag type #id# text	Changes NDEF tag of supplied ID to be a text tag	
nfc tag type #id# URL	Changes NDEF tag of supplied ID to be a URL	
nfc tag type #id# WApp	Changes NDEF tag of supplied ID to be a Windows Application Launcher	
nfc tag type #id# AApp	Changes NDEF tag of supplied ID to be an Android Application Launcher	
nfc tag payload #id# #data#	Updates the payload of NDEF tag of supplied ID to the supplied data	
nfc list	Lists the NDEF message types and data payloads	
nfc space	Shows the total free/used space of the committed NDEF tags	NFCNdefMsgGetInfo()

nfc commit	Commits the current list of NDEF messages to the NFC stack	NFCFieldSense()
		NFCNdefMsgNew()
		NFCNdefRecAddGeneric()
		NFCNdefMsgCommit()
		NFCNDefMsgDelete()

nfc clear	Clears all NDEF messages and values (does not commit change to stack)	
nfc events 0	Disables NFC message outputting	OnEvent EVNFC disable
nfc events 1	Enables NFC message outputting	OnEvent EVNFC call HandlerNfc
deepsleep	Puts module into deep-sleep mode (if NFC has been enabled and field sense is on then the device will awaken when an NFC field is detected).	SystemStateSet()
quit	Exit the application	
exit	Exit the application	

References

The following documents are also accessible from the BL652 product page of the Laird website (Documentation tab):

- BL652 *smart*BASIC extension manual
- BL652 Datasheet
- UwTerminalX

REVISION HISTORY

Version	Date	Notes	Approver
1.0	6 Sept 2016	Initial Release	Jonathan Kaye