Near Field Communications Manager: BL654 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 BL654 module and show how to utilise this functionality with the NFC Manager *smart*BASIC application by testing the functionality over the UART and introducing ideas for implementing NFC in designs.

Potential Use-Cases for NFC on BL654

The following are potential use cases for NFC on the BL654

- **Embedded advertising** – NFC can be used to interact with customers by placing tags at bus stops or convention areas which can open websites when tapped. Rather than a read-only disposable tag which must be replaced for each data update, you can leverage the Bluetooth Low Energy (BLE) functionality of the BL654 to update the tag data and to only allow connections from trusted devices. This ensures that the tags cannot be erased or modified by unauthorized 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 BL654 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 BL654 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 BL654 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 BL654 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 utilize NFC in the same way.

In this sample application, the BL654 is 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. Please note that Apple iOS devices with NFC chipsets currently only support detecting NFC tags and reading messages that contain NDEF messages. At the time of this writing, Apple IOS devices do not support writing to external NFC tags.

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 BL654 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 BL654, please read the NFC section in the BL654 *smart*BASIC manual which has a very detailed description on how NFC works and how it is implemented in the BL654 module.

Requirements

- Laird DVK-BL654, Part # 455-00001 or 455-00002
- NFC film antenna
- 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/BL654-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, see the *BL654 smartBASIC Extension Manual*. You can access it from the BL654 product page in the Documentation section: https://www.lairdtech.com/products/bl654-ble-thread-nfc-modules

Like other *smart*BASIC functionality, NFC can be controlled on the BL654 in a state-machine based system with three distinct segments (Figure 1):

Setting up NFC	The NFC hardware is initialized, the NDEF messages are created and committed, and then the NFC field is enabled. Whenever a device reads the BL654 NFC tag, an optional *smart*BASIC event is thrown which indicates that the NFC field is established, broken, or that reading the tag is finished.
Changing NFC data	The data on the NFC tag is currently read-only to remote devices. The data can be changed from the BL654 by disabling the NFC field, deleting the NDEF messages, creating the new messages, committing them to the stack, and then finally enabling the NFC field.
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 reinitialized 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 must be deleted. Resetting the memory only clears the data so that it does not have to be reallocated to be re-used by an NFC message.

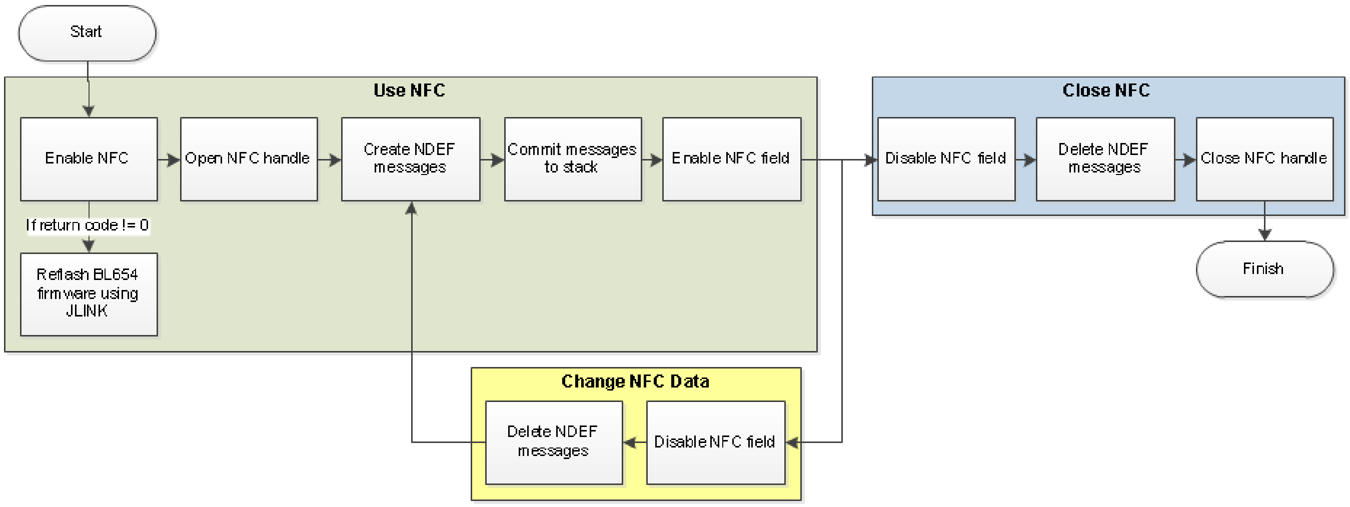


Figure 1: NFC in smartBASIC overview

BL654 Development Kit Setup

To set up the development kit, follow these steps:

1. Configure the BL654 development kit with the following settings:
   - Power source switch (SW4) to USB_5V
   - Switches SW6 and SW5 to 3V3
   - Set SW7 in the middle position (Vdd) and SW11 to the right, as shown in Figure 2.

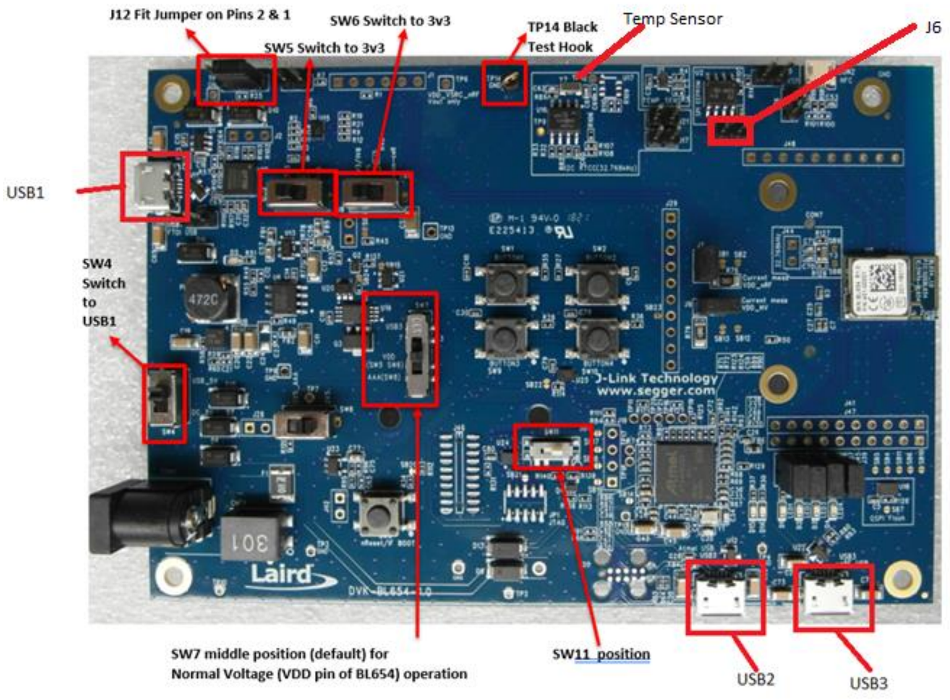


Figure 2: BL654 development board

2. Insert the film antenna into CON2 on the BL654 Development kit (see Figure 13).
3. Connect your BL654 Development kit to your PC via the USB Micro cable at the USB1 port. The power LED illuminates when the board is receiving power.
4. In most cases, this should be a plug-and-play operation, but if asked, install the FTDI USB-to-Serial driver (found at http://www.ftdichip.com/FTDrivers.htm).
5. Open UwTerminalX.
6. In the Config tab, select **BL65x** from the device list drop-down.
7. Select the serial port associated with your development board. If your version of UwTerminalX does not have the BL65x dropdown then you should update to a newer version or manually use the settings shown in Figure 3.

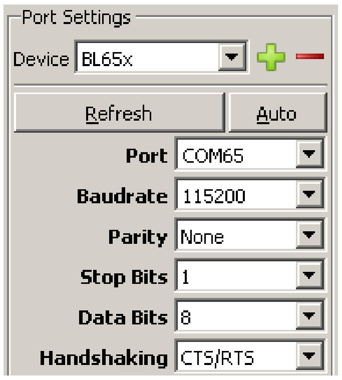
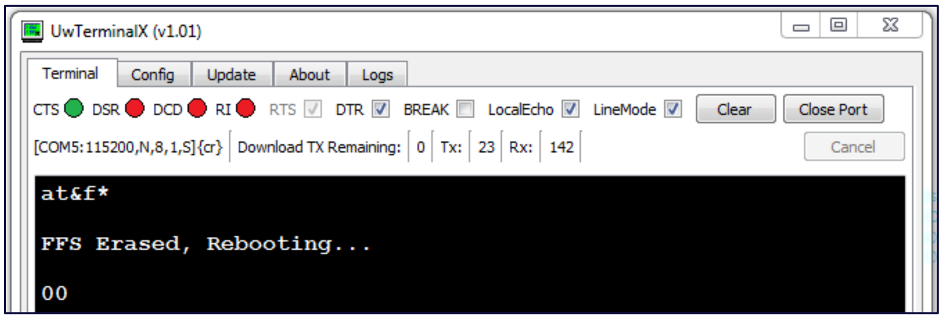

Port Settings
Device: BL65x
Port: COM65
Baudrate: 115200
Parity: None
Stop Bits: 1
Data Bits: 8
Handshaking: CTS/RTS

Figure 3: Setting connection parameters

8. Click **OK** to advance to the Terminal tab.
9. Use UwTerminalX to return the BL654 to factory defaults using the command *at&f** as shown (Figure 4). 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.



[COM5:115200,N,8,1,S]{cr} Download TX Remaining: 0 Tx: 23 Rx: 142

```
at&f*

FFS Erased, Rebooting...

00
```

Figure 4: Factory default

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

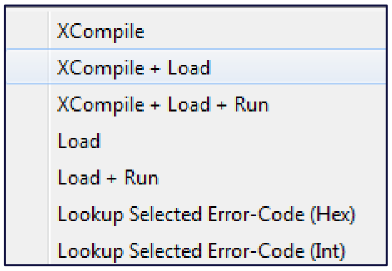

XCompile
XCompile + Load
XCompile + Load + Run
Load
Load + Run
Lookup Selected Error-Code (Hex)
Lookup Selected Error-Code (Int)

Figure 5: XCompile Load and Run

11. 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/BL654-Applications/master/Applications/nfc.manager.sb.
12. Wait for the NFC manager program to load; this should take approximately 10 seconds.

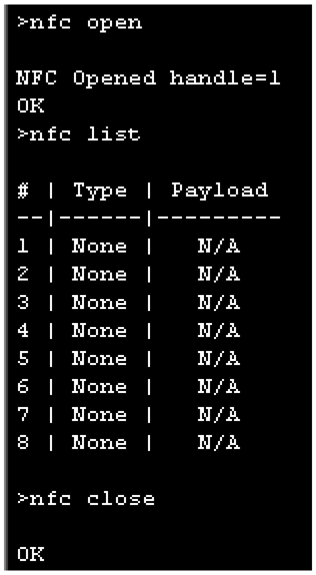
    NFC Manager can now be run by typing *nfc* followed by return.

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 – the NFC device is not open. To perform any NFC operations, the NFC sub-system must be opened which returns a handle on which to perform operations. Do this by sending the command *nfc open* over the UART. Because the BL654 only has a single NFC device, the returned handle is always 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 6).



```
>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 6: Basic NFC functions

Table 1: Basic NFC functions

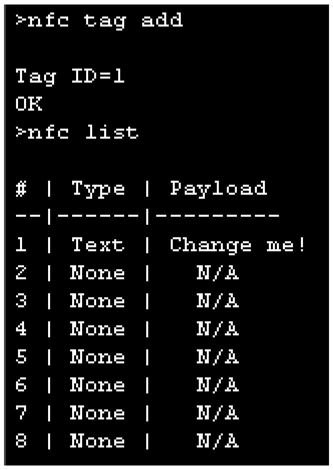
Command	Description	*smart*BASIC Command
nfc open	Opens the NFC sub-system hardware	NFCOpen()
nfc close	Closes the NFC sub-system hardware	NFCClose()
nfc list	Lists the NDEF message types and data payloads	

CREATING NDEF MESSAGES

To create NDEF messages, follow these steps:

1. Ensure that the NFC device is open using *nfc open.*
2. Use *nfc tag add* to add a new tag. Using this application, you can create up to eight NDEF messages.

   Once a new message is added, the tag ID is returned. View the new tag by using the command *nfc list* (Figure 7).

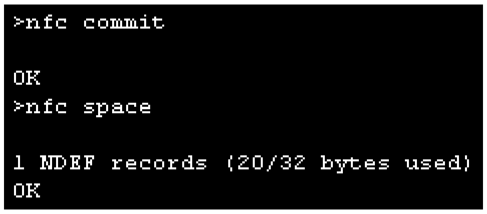


```
>nfc tag add

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 7: Adding a NDEF message

   At this stage the NDEF message is not committed.

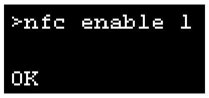
3. Commit the data to the stack using *nfc commit*.
4. With the data committed, you can use *nfc space* to retrieve information about the NDEF records. It displays the total space used and allocated for the NFC tags in bytes (Figure 8).



```
>nfc commit

OK
>nfc space

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

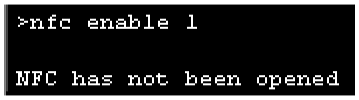
Figure 8: Commiting the NFC tag and vieiwing the space used

5. Enable the NFC field sense using *nfc enable 1*. Once enabled, the data in the tag becomes read-only. It cannot be changed from *smart*BASIC until NFC is disabled. To change the data, disable the field sense by using *nfc enable 0*. Once changed, re-enable it using *nfc enable1* (Figure 9 – Figure 11).



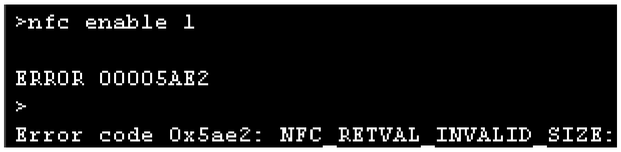
```
>nfc enable 1

OK
```

Figure 9: NFC field enabled



```
>nfc enable 1

NFC has not been opened
```

Figure 10: Error if NFC isn't open



```
>nfc enable 1

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

Figure 11: Error if no NDEF tags have been added

Table 2: Creating NDEF messages commands

Command	Description	*smart*BASIC 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 Google Nexus 6 was used so the exact steps for using NFC functionality for your device may differ. Please check with your device manual for information on how to configure and use NFC.

To read NFC tags on Android, follow these steps:

1. On the phone, enable NFC functionality from the settings (Figure 12).
2. Hold your mobile device flush against the BL654 NFC antenna (the range of NFC is very small - Figure 13) to allow the device to read the NFC tag.

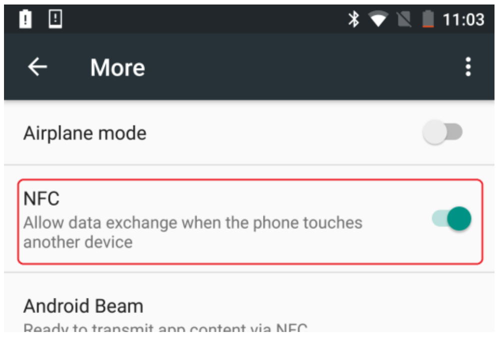

11:03
More
Airplane mode
NFC
Allow data exchange when the phone touches another device
Android Beam
Ready to transmit app content via NFC

Figure 12: Enabling NFC on Android

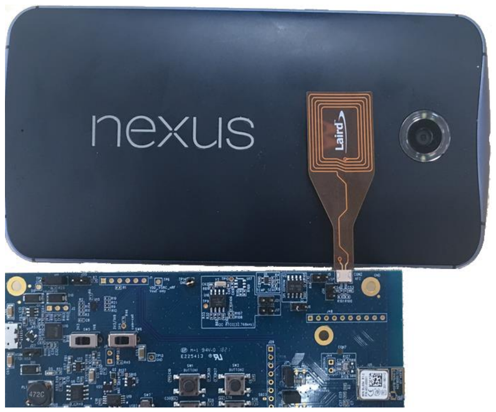
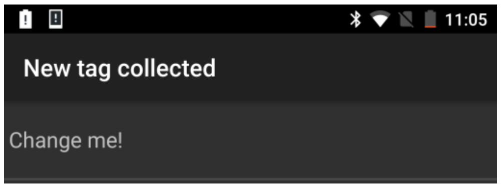


Figure 13: BL654 NFC antenna held flush against android device

The tag currently contains a simple text message which should be displayed on the screen (Figure 14).



11:05

New tag collected

Change me!

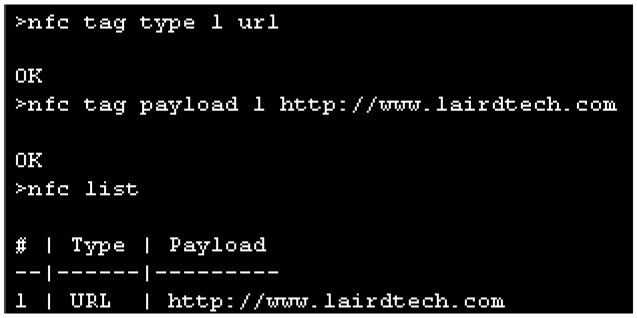
Figure 14: Default text tag as detected on Android

Launching URLs in a Web Browser

The tag type can be changed to advertise a URL instead of only text. Some devices automatically open a web browser on the page indicated by the tag while other devices ask the user for a confirmation before visiting the site.

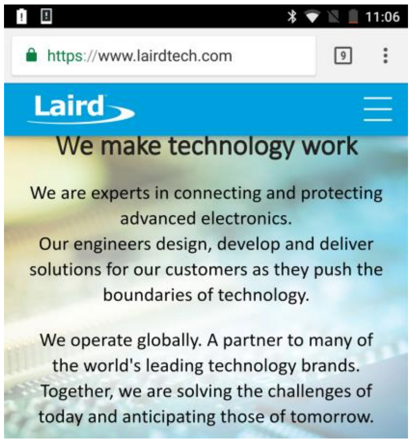
1. Send the command *nfc tag type 1 url* to change the tag type to a URL. Note that *1* represents the tag ID. Because you only added one tag, *1* is currently the only valid tag.
2. Set the URL using the command *nfc tag payload 1 http://www.lairdtech.com* (Figure 15).
3. Update the NDEF message using the command *nfc commit*.

   Now, if you hold your phone to the tag, it should greet you with an option to open the URL or it may direct you to the URL automatically (Figure 16).



```
>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 15: Changing the NDEF message to a URL and setting the URL



https://www.lairdtech.com

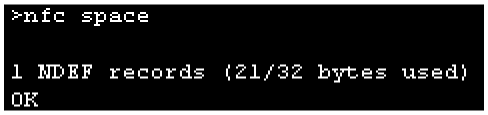
We make technology work

We are experts in connecting and protecting advanced electronics.
Our engineers design, develop and deliver solutions for our customers as they push the boundaries of technology.

We operate globally. A partner to many of the world's leading technology brands. Together, we are solving the challenges of today and anticipating those of tomorrow.

Figure 16: Website opened via URL NDEF Message

Issue the command *nfc space* to check the space used by the NDEF messages in the stack (Figure 17). The displayed size is only 21 bytes but the length of the URL itself is 24 bytes long.



```
>nfc space

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

Figure 17: NDEF URL message size

The reason that the entire NDEF message with a URL is shorter than the URL itself is because the NDEF URI (Uniform Resource Identifier) format has the first byte of the NDEF URI message indicate the start protocol and address. This saves space for embedded devices and requires less memory. Inside the NFC Manager application source code, the process of shortening the URI is done in the *CompressURL$* function. Some of the most common protocols are listed in the following table:

Table 3: NDEF URI common protocols

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* which saves ten bytes. This URI shortening system is part of the NDEF specifications and is universally supported.

Table 4: Commands for launching URLs in a web browser

Command	Description	*smart*BASIC 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 to automatically 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*.

1. Change the original NDEF message to an Android application launcher by sending *nfc tag type 1 AApp*.
2. Set the application to open by using *nfc tag payload 1 com.lairdtech.lairdtoolkit*.
3. Use *nfc commit* to commit it.

   Now, when you tap an Android device onto the NFC antenna, the Laird Toolkit application opens (if installed) (Figure 18). If not installed, the Google Play store opens and displays the application page.

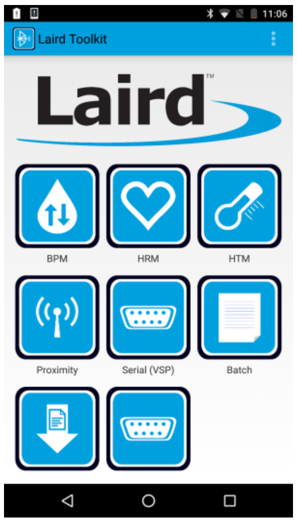


Figure 18: Laird Toolkit launched via Android app launcher NDEF message

Table 5: Commands for launching Android applications

Command	Description	*smart*BASIC 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	

Command	Description	*smart*BASIC Command

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. For example, you 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.

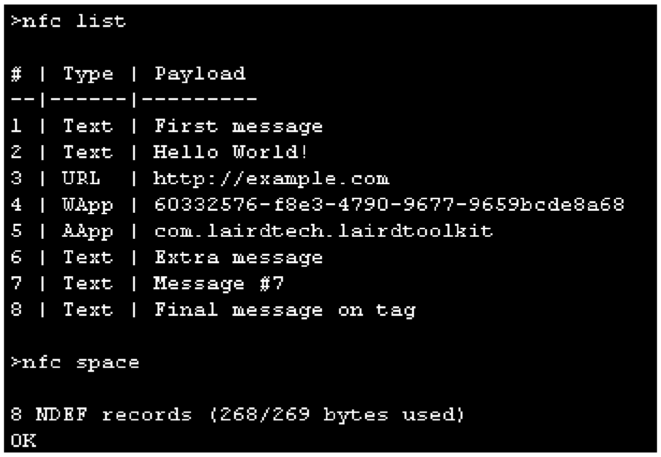
To create multiple messages using the NFC Manager, issue the command *nfc tag add*. This adds a new message and returns the handle for the tag.

Note: Up to eight messages can be created with the NFC manager application, but this is not a hardware limit. You can change this, if necessary by using a define in the application source code (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	A number between 1-8
#type	Text, URL, AApp, or WApp
#data	The payload for the message

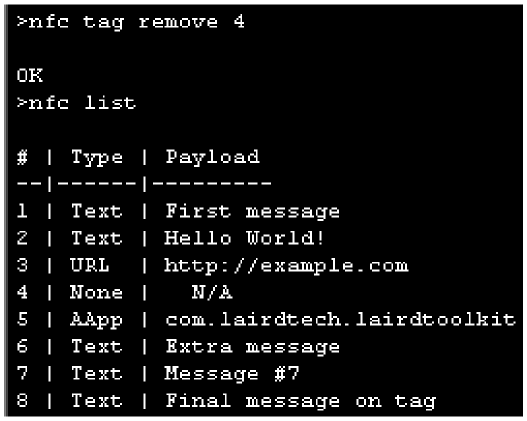
A full NFC tag with eight messages is shown in Figure 19.



```
>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
```

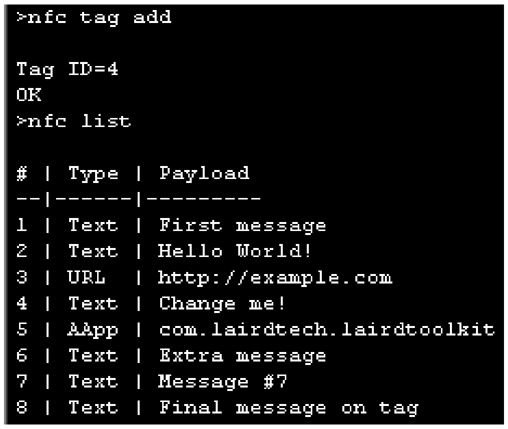
Figure 19: Eight-message NFC tag

Messages can be deleted if they are no longer needed using *nfc tag remove #id* (Figure 20). The order of NFC messages does not change when a message is deleted. If required, the tag can be re-added using *nfc tag add* with the default type and value (Figure 21). Remember, once the messages are changed, these changes do not appear on the tag until you issue the *nfc commit* command.



```
>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 20: 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 21: Re-adding a tag after deleting

Table 6: Commands for multiple NDEF messages

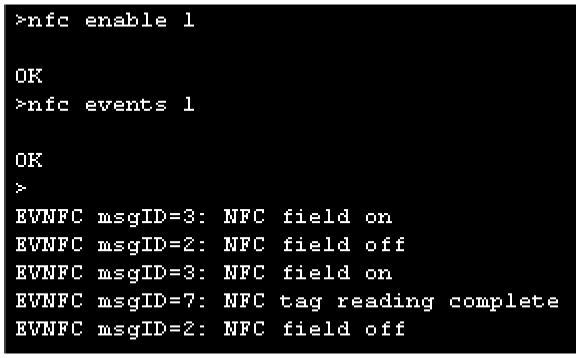
Command	Description	*smart*BASIC Command
nfc tag add	Adds an NDEF tag and returns the ID	
nfc tag remove #id#	Removes NDEF tag with supplied ID	
nfc tag type #id# text	Changes NDEF tag of supplied ID to be a text tag	
nfc tag type #id# URL	Changes NDEF tag of supplied ID to be a URL	
nfc tag type #id# 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 22).



```
>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 22: NFC event messages when enabled

Table 7: Commands for NFC status messages

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

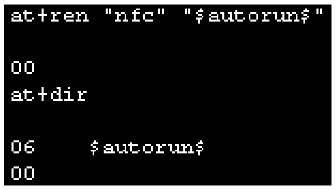
Using NFC to Awaken BL654 in Deep Sleep Mode

NFC can be used to awaken a BL654 that is in Deep Sleep mode (a very low power mode). For this to work, NFC must be set up and enabled, with the application renamed as an autorun application.

To use NFC to awaken the BL654 in deep sleep mode, follow these steps:

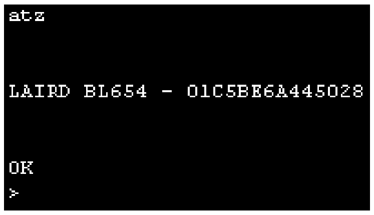
1. Reset the BL654, either by enabling and disabling BREAK from UwTerminalX or by pressing the reset button on the development board.
2. Rename the application so that it automatically runs on start-up by issuing the command *at+ren "nfc" "$autorun$"*.
3. Confirm that the rename was successful by sending *at+dir* (Figure 23).
4. Ensure that autorun mode is enabled.
    a. Place the jumper on J12 of the development board on pins 1-2.
    b. Check **DTR** in UwTerminalX.
5. Send the command *atz*.

   The NFC Manager application should start (Figure 24).



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

00
at+dir

06     $autorun$
00
```

Figure 23: Renaming NFC Manager as an autorun application



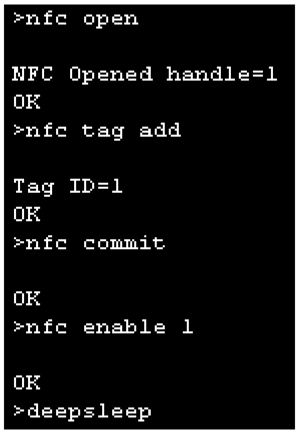
```
atz


LAIRD BL654 - 01C5BE6A445028


OK
>
```

Figure 24: 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 must be created.

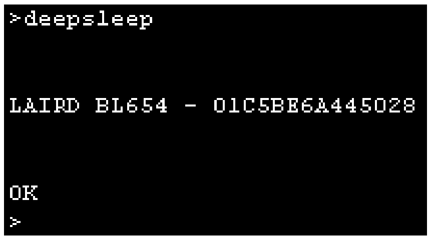
6. Open NFC with *nfc open*.
7. Using *nfc tag add,* add a new sample tag.
8. Using *nfc commit*, commit to the stack.
9. Enable the NFC field with *nfc enable 1*.
10. Put the BL654 module into deep sleep mode by sending *deepsleep* (Figure 25).

    The device is now in its lowest power state and does not respond to any UART commands.
11. Restart the device by holding an NFC reader over the NFC antenna. The module resets and executes the autorun application (NFC Manager) (Figure 26).



```
>nfc open

NFC Opened handle=1
OK
>nfc tag add

Tag ID=1
OK
>nfc commit

OK
>nfc enable 1

OK
>deepsleep
```

Figure 25: Setting up BL654 for NFC deep sleep waking



```
>deepsleep


LAIRD BL654 - 01C5BE6A445028


OK
>
```

Figure 26: BL654 woken up from NFC field

To exit out from the autorun application, send *exit* or *quit* to return to command mode. You can remove the autorun application by deleting the file using *at+del "$autorun$"* or by clearing the module and all non-volatile data by using *at&f**.

Table 8: Commands for Using NFC to awaken BL654 in deep sleep mode

Command	Description	*smart*BASIC Command
at+ren "nfc" "$autorun$"	(From command mode) Renames the application *nfc* to *$autorun$* so that it is 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()

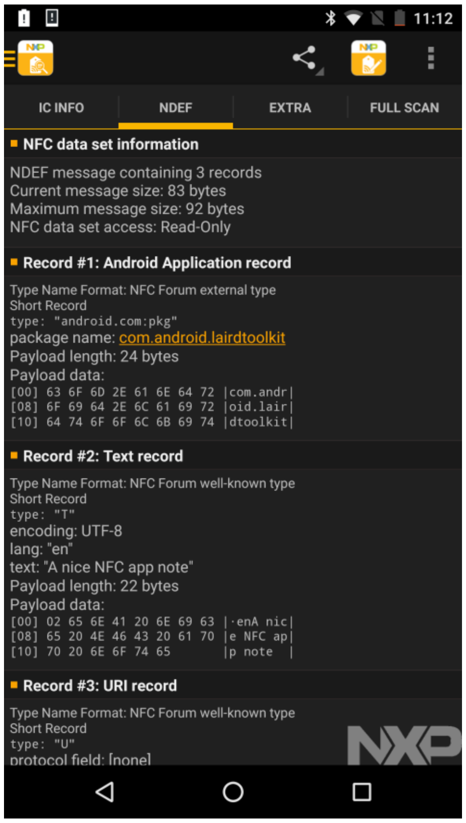
deepsleep	Puts the module into deep-sleep mode If NFC has been enabled and field sense is on, then the device will awaken when an NFC field is detected.	SystemStateSet()

Command	Description	*smart*BASIC Command
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	Exits the application	
exit	Exits the application	

Additional NFC Tag Details and Debugging

It may be necessary to debug NFC tags or view the raw data being sent. NFC TagInfo by NXP Semiconductor is a useful application for this. It's available from the Google Play app store:

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 27).



```
NFC data set information
NDEF message containing 3 records
Current message size: 83 bytes
Maximum message size: 92 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.android.lairdtoolkit
Payload length: 24 bytes
Payload data:
[00] 63 6F 6D 2E 61 6E 64 72 |com.andr|
[08] 6F 69 64 2E 6C 61 69 72 |oid.lair|
[10] 64 74 6F 6F 6C 6B 69 74 |dtoolkit|

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 Forum well-known type
Short Record
type: "U"
protocol field: [none]
```

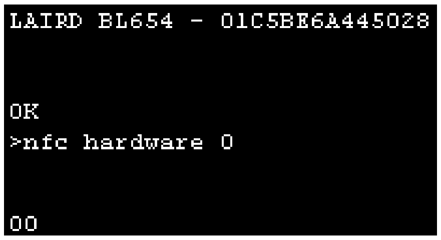
Figure 27: NXP NFC TagInfo

Disabling NFC Functionality in the BL654

When NFC functionality is enabled, which is by default, two IO pins on the BL654 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* using the NFC Manager application (Figure 28).

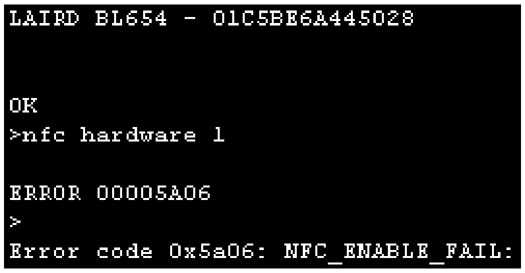
Note: Once the NFC pin functionality is disabled, it can only be re-enabled by flashing the BL654 firmware using the JLINK interface (a UART upgrade does not work). See the *BL654 smartBASIC Extension Manual* for further details. You can access it from the BL654 product page in the Documentation section: https://www.lairdtech.com/products/bl654-ble-thread-nfc-modules

If NFC functionality is disabled, this application does not work. An error is returned when an attempt is made to use the NFC *smart*BASIC functions (Figure 29).



```
LAIRD BL654 - 01C5BE6A445028


OK
>nfc hardware 0


00
```

Figure 28: Disabling NFC functionality



```
LAIRD BL654 - 01C5BE6A445028


OK
>nfc hardware 1

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

Figure 29: Error returned when NFC is disabled

Table 9: Commands for disabling NFC functionality

Command	Description	*smart*BASIC Command
nfc	Runs the loaded NFC Manager program	
nfc hardware 0	Disables the NFC hardware and resets the module	NFCHardwareState()
nfc hardware 1	Returns *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; 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 utilization. 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 is committed then this tag is made active and the previous tag (if there was one) is deleted.

Full List of NFC Manager Commands

Table 10 contains all the NFC commands available in the **NFC Manager** application.

Table 10: NFC Manager commands

Command	Description	*smart*BASIC Command
nfc open	Opens the NFC sub-system hardware	NFCOpen()
nfc close	Closes the NFC sub-system hardware	NFCClose()
nfc hardware 0	Disables the NFC hardware and resets the module (requires JLINK flash to re-enable)	NFCHardwareState()
nfc hardware 1	Returns 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 ()
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 is enabled and field sense is on, then the device awakens when an NFC field is detected.	SystemStateSet()
quit	Exit the application	
exit	Exit the application	

References

The following documents are also accessible from the BL654 product page of the Laird website (Documentation tab):

- BL654 *smart*BASIC extension manual
- BL654 Datasheet
- UwTerminalX

Revision History

Version	Date	Notes	Contributor(s)	Approver
1.0	18 July 2018	Initial Release	Jamie McCrae	Jonathan Kaye