

!d hand

developer guide



version 1.1 - june 2011

nedap
R E T A I L

Table of contents

1. Introduction.....	4
Features.....	4
Manual	6
Support	6
2. Choosing your development platform.....	7
iOS platform (Apple inc.).....	7
Windows CE/Mobile (Microsoft)	8
Android (Google as leading partner).....	8
Other platforms	8
3. Structure of the API	9
Tag Data Translation	9
The layers of the API	9
Choosing the right layer	9
Data objects	9
4. Tag Data Translation.....	11
5. Presentation layer.....	12
Objects.....	14
General.....	15
User interaction	17
RFID.....	19
Barcode.....	23
NFC cards.....	25
6. Session layer.....	26
Management.....	26
General.....	26
User interaction	29
RFID.....	30
Barcode.....	35

NFC.....	36
7. Transport layer	38
Packet structure	38
Commands to !D Hand	39
Responses from !D Hand	51
8. Getting started with development.....	58
iOS (in Xcode 4.0)	58
Windows (in Visual Studio)	62
A. Frequently Asked Questions.....	66
B. Update firmware	67
C. Licensing	72
D. Change notes	73

1. Introduction

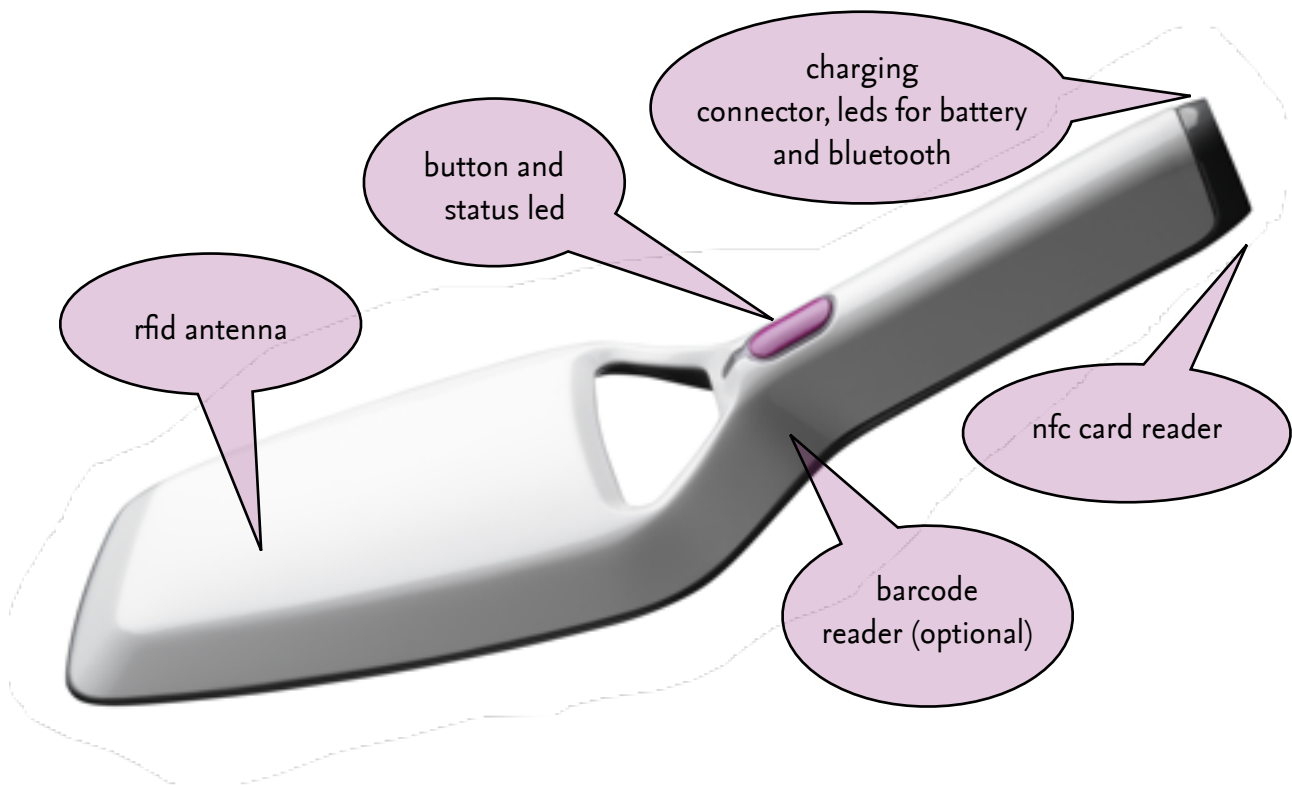
The !D Hand is the first next-generation RFID handheld reader. Design and easy of use are the most important properties of the device - not only for the end user, but also for the developer. Regarding features, we provide excellent RFID reading facilities, optional barcode reading and NFC card reading for security purposes. A long battery life and easy user interaction is also guaranteed.

This guide will explain you how to make software that works together with the !D Hand, to make the best integrated RFID solutions possible. This guide will assume basic knowledge of RFID if you use the presentation layer API, but if you use the session layer API you'd probably need more knowledge. An excellent book on UHF passive RFID is "The RF in RFID: Passive UHF RFID in Practice", from Daniel M. Dobkin - available on Amazon¹.

Features

rfid	UHF RFID: EPC Class 1 Generation 2 support all common operations (read, write, lock, kill) cross-dipole antenna output power around 25 dBm (reader output power 25 dBm) reading distance 1 - 1.5m, with typical retail tags
barcode (optional)	1D laser barcode scanner reads all standard barcode types
near field communication	reader/writer functionality compatible to ISO/IEC 14443 A&B, MIFARE, FeliCa and NFC Forum tag types (MIFARE Ultralight, Topaz, FeliCa, MIFARE DESFire)
user interaction	one button, one status led accelerometer, shake detect vibration motor beeper
bluetooth	bluetooth 2.1+EDR, Class 2 supports Secure Simple Pairing
API	high-level API for iOS and Windows CE/Mobile (supporting inventory, write and verify, monitoring, etc.) - other platforms can be added on special request communication protocol documentation for any Bluetooth-enabled device
battery	re-chargeable Li-ion battery with 2600 mAh capacity battery lasts for around 4-6 hours during inventory standby-time is four weeks charge connector is micro USB-B charge with 5V DC, between 500 mA and 1.5A, both PC-USB and charger-USB
weight	250 gram
size	11 cm (width) 5 cm (height) 32 cm (length)

¹ <http://www.amazon.com/RF-RFID-Passive-UHF-Practice/dp/0750682094/>



RFID reading

RFID reading is implemented according to the EPC Global Class 1 Generation 2 standard (ISO 18000-6C). All standard operations are implemented (read, write, lock, kill). Upon request, chip-vendor specific commands (like NXP's EAS bit) can be implemented. Furthermore, it's possible to set most of the parameters that are defined in the specification.

For now, only the European ETSI frequency range (EN 302 208) is supported, but in the near future all other regions will be added (North America, China, Korea, Japan, Australia, etc.) - depending on market demand.

The included antenna is a cross-dipole antenna, automatically switching between both orientations while reading and writing to tags.

Barcode reading (optional component)

The barcode reader is an optional component. All standard types of barcodes are recognized, furthermore it's possible to set the length of the barcodes that the reader will decode.

NFC card reading

There is a NFC card reader in the grip of the !D Hand that's capable of reading ISO/IEC 14443 type A cards. For now, it's only possible to read the card serial number - which should be enough for most basic authentication procedures. Advanced operations (reading sector data) and other types of cards (ISO/IEC 14443 type B) are supported in hardware, and will be implemented in software on request.

User interaction

There are several methods to interact with the user. These are:

- LED. It's possible to have the LED blinking or not.
- Button. The user can press the button on the !D Hand to start an action
- Sound. The !D Hand can produce several pre-defined sounds.
- Vibration. The !D Hand has a vibration motor inside.
- Accelerometer. By shaking the !D Hand, the user can generate an action.

Bluetooth

The !D Hand is equipped with Bluetooth version 2.1+EDR. It supports Secure Simple Pairing (for pairing without any pin codes or user interaction), and is a class 1 device (approximately 10 meters range).

Charging

Charging can be done using the micro USB connector. It is recommend to charge using the accompanied multi-head micro USB charger, but it's also possible to use an standard USB outlet on any computer around. Please note that charging with a dedicated charger is a lot faster than charging on a standard USB outlet of a computer.

Manual

This section is a brief overview of how the !D Hand operates in a normal situation. A graphical version of this manual is included in the box.

Powering on and off

To enable the !D Hand, press the button. The status LED will turn on. After five minutes of no activity (meaning, no bluetooth connection), the !D Hand will automatically turn off.

If the button is pressed, and the !D Hand beeps, the battery is empty. Please charge the !D Hand before using it.

To reset the !D Hand, hold the button for more than ten seconds. This can also be used to force a disconnect from a device.

Status LED

The status LED indicates if the !D Hand is currently doing something. If the LED is blinking, it's operating. If it's just on, the !D Hand is on, but doing nothing. If the LED is off, the !D Hand is off.

Bluetooth LED

The Bluetooth LED indicates whether there is a Bluetooth connection or not. If the LED is off, there is no connection. If the LED is blinking, the connection is in 'progress' and if the LED is on, there is a Bluetooth connection.

Battery LED

If the battery LED is off, the status is normal. It'll be orange when it's charging, and green when the charging is done. The LED will blink red if the battery is almost empty (<10 %) and will burn continuously red when the battery is empty.

Support

To provide you with the optimal support for the !D Hand, there are two basic methods to get started:

1. For all questions from end-users about the operation of the device, RMA's, ordering, etc. please contact the Nedap Retail business partner where you've obtained the device. If you've received the device directly from us, please find the business partner that operates in your country at www.nedap-retail.com. If there's no business partner in your country, please use option 2 to contact Nedap.
2. For all questions regarding development, bugs, etc. you can contact the headquarters of Nedap Retail directly. You can e-mail the Store !D team at rfidretail@nedap.com, or call at +31544471111. A ticket will be filed, and you'll be kept posted about updates.

2. Choosing your development platform

The development platform is one of the most important decisions you've to make. This choice determines for a part how easy the end user can work with your device (and thus the amount of support requests you'll get), but also how easy it is to make a roll-out and manage all the devices. For each of the platforms we've thought about, I'll list some things you can think about to make an informed choice and decision.

Please remember that in principal the !D Hand is platform agnostic. It's possible to use it with any mobile device (or fixed device) that has a Bluetooth interface.

iOS platform (Apple inc.)

iOS started as a new mobile platform when the first iPhone was introduced in 2007, but came into real play with the introduction of version 2.0 - supporting the iPhone 3G and enabling app development by 3rd parties. iOS is known for its very simple user interface and interaction (capacitive touchscreen, no need for stylus) and stability (Unix basis).

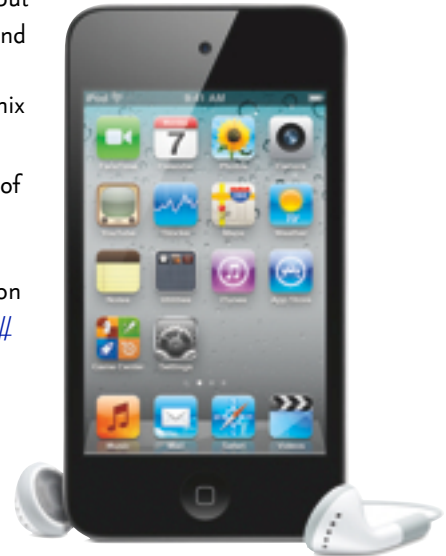
Initially, the effort was focussed on the consumer market, but with the introductions of versions 3.0 and 4.0 (and accompanying devices), more and more business-needed features were implemented. These features include device encryption, feature lockdown, remote wipe, mobile device management, etc. etc. For more information on these business features, please see <http://www.apple.com/iphone/business/> and <http://www.apple.com/ipad/business/>.

The fact that iOS is a closed and locked environment both offers advantages and disadvantages. We strongly believe the advantages (sand-boxed and signed apps, no viruses or tampering possible) outweigh the disadvantages (less straightforward app distribution). For more information see <http://www.apple.com/iphone/business/integration/>.

The devices that run iOS (iPod, iPhone, iPad) have a very predictable release cycle (one version each year) and have shown very good backwards capabilities (new apps still run on the original iPhone). They're also very cheap, with an iPod starting from EUR 229,- (including VAT, and excluding possible volume discounts).

For the distribution of applications, there are basically two options:

1. App Store. The App Store is the most known way to distribute app. Apple has to approve your app, and afterwards it's available for everyone to download. Either for free, or for a fixed price (of which Apple takes 30% for distribution). Please see <http://developer.apple.com/programs/ios/> for information.
2. In-house distribution. For in-house, proprietary apps it's possible to distribute them without approval from Apple and without paying Apple for distribution. You've to set up your own distribution server, or can use a service from a 3rd party to do that. Please see <http://developer.apple.com/programs/ios/enterprise/> for information.



Windows CE/Mobile (Microsoft)

Windows CE/Mobile is the oldest mobile platform supported by the !D Hand's API. (Windows Phone 7 is for the moment not taken into account here, since there is no documented accessory program at the moment). Windows CE/Mobile has a quite large installed base in business mobile devices, and a large range of devices that run it, but it has some big disadvantages:

- The user interface is out of the 90's era. Devices still use resistive touch screens that need a stylus to operate. Furthermore, interface elements are not clear to users and screens tend to be overcrowded. Furthermore, changing a simple setting is too difficult for users.
- The devices itself are rugged and complicated, most devices tend to have a lot of buttons. Users are not well used to these kind of devices (in contrary to iOS devices), so they'll need a lot of expensive training to get to understand them.
- Making apps is more difficult than for other platforms. API's are less sophisticated and there is no standard way of distributing apps.
- There is no standard way to connect with an accessory. Each device maker (and sometimes each device) has a different Bluetooth stack requiring a different implementation on how to connect with an external device.

But, because the usage of Windows CE/Phone is quite widespread, we still decided we wanted to support it in our API. Because of the not-unified way of connecting and pairing with Bluetooth accessories, that part is left out for the developer. The following devices were tested during development (and are at least guaranteed to work).

- Motorola ES400 (Windows Mobile 6.5)
- Motorola MC55 (Windows Mobile 6.1)
- NordicID Morphic (Windows CE 6.0)

If you've any other device that you'll want to use, and if you face any issues: please let us know. We're here to help you.



Android (Google as leading partner)

Android is a relatively new and - although very promising - not recommend to use in a business environment at the moment. There are basically two reasons for that:

- Android is not always Android. There is a common base, but there are a lot of variations (because the base is open source) and versions around.
- Devices are not released in a timely and predictable way and available for a longer time. Furthermore, there are mostly just phones, not devices without a cellular modem. The phones are mostly more expensive.

But, some other things are really interesting:

- It is possible to use an hardware accessory connected via Bluetooth; although it is not as advanced as with the iOS accessory protocol.
- App distribution is a lot more straightforward; it is easier to set up your own app store.

We expect the disadvantages to fade in 2011 and companies will introduce some first Android business devices. At the moment, there is no API for Android, but we'll watch it closely. If you want, you can use the Transport Layer documentation to build your own API. We'd be happy to help you with that, even with providing the source of our iOS/Windows API's.



Other platforms

Like we said before, every platform with a Bluetooth adaptor that supports the serial port protocol can use the !D Hand. You can then think about legacy devices, but also Blackberry's (maybe even the new Playbook?).

3. Structure of the API

This chapter will explain how the API is constructed, and how to choose which layer to use for your applications' development.

Tag Data Translation

Tag Data Translation is used to convert the possible data representations in the EPC standard from one to another (for example, from binary to pure identity URI). This library also contains objects for EPC tags and EPC observations.

The layers of the API

Bluetooth

The Bluetooth layer is as in the Bluetooth 2.1+EDR standard, the serial port protocol. The name of the device is '!D Hand XXX', where XXX denotes the last three characters of the serial number, that are depicted on the bottom of the !D Hand. This number can be used to verify which !D Hand to connect to; if you've multiple.

Transport layer

The transport layer implements a binary protocol that flows over the Bluetooth serial port. You'll never have to touch this layer, unless you want to develop on an OS that's not supported by the session or presentation layer.

Session layer

The session layer acts as a wrapper around the transport layer. It provides the same functionality, but enables simple function calls to do the communication. You can for example set all the parameters for RFID reading (session, power, Q, M-value, etc.), and do a single RFID read cycle. Also, it's possible to set the LED to blinking or have the !D Hand vibrate for a while.

The session layer can be used for more advanced, custom and complex operations that are not currently supported by the presentation layer.

Presentation layer

The presentation layer is the most sophisticated layer, building on top of the session layer. This layer provides more sophisticated functionality, like 'inventory' (that does continuous reading, automatically drives the led and beeps when a 'new' tag is detected) or 'write EPC' (checking if there's only one tag in the field, writes to it and then verifies what's written... sounding a beep if everything was successful).

Choosing the right layer

The developer has to make a decision which layer to use to connect to the !D Hand: the transport layer, the session layer or the presentation layer.

We recommend to use the presentation layer as much as possible, but it might be that you'd like to have an advanced feature that's not possible with the presentation layer. Since it's not possible at the moment to use the session API and the presentation API at the same time, a decision has to be made.

We plan to add more features to the presentation layer in the coming months, so if you feel there's something missing that should be there, please let us know. Thing that we're currently considering are:

- Password management (locking a tag while writing)
- Read and write data from the user memory

Data objects

In iOS and Windows CE data objects are defined differently. The following table lists the naming differences.

named here	type on iOS	type on Windows CE
String	NSString	String
Data	NSData	byte[], accompanied by a length in int
Bool	BOOL	bool
Integer	NSInteger or NSNumber	int
Select	NHSelect	Select
Date	NSDate	DateTime
Array	NSArray	Collection<...>
EPC observation	EPCObservation	EpcObservation
EPC code	EPCCode	EpcCode

4. Tag Data Translation

Tag Data Translation is based on the official specifications from EPC global² (GS1), and implemented roughly based on Fosstrak's implementation³. The translation capabilities can be directly accessed by using the EPCTDTEngine object, or indirectly (and in most cases more convenient) by using the properties of the EPCCode object.

The concept behind the EPCCode object, is that it's initialized either with binary data from an RFID tag, with a string representing hexadecimal characters or one of the other representations or with a GTIN14, a serial number and company prefix lengths. Other representations then can be obtained by using the right property.

Objects

EPC code

The EPC code is used to represent an EPC code.

property	type	description
binaryData	Data	data containing the EPC code in byte format
binaryString	String	string containing the BINARY representation
hexString	String	string containing the BINARY representation, in hexadecimal format
tagEncoding	String	string containing the TAG_ENCODING representation
pureIdentityURI	String	string containing the PURE_IDENTITY_URI representation
legacy	String	string containing the LEGACY representation
onsHostname	String	string containing the ONS_HOSTNAME representation

² <http://www.gs1.org/gsmp/kc/epcglobal/tdt/>

³ <http://www.fosstrak.org/>

5. Presentation layer

The presentation layer is implemented asynchronously. A called function is immediately returned. The actual information is passed back via `NSNotificationCenter` (iOS) or events (Windows).

In the presentation layer, there are two type of operations:

1. Continuing operation, for example inventory. The user presses the button to start the operation, and has to push the button again to stop the operation. During this time, the status LED is blinking, and continue and pause notifications are sent to indicate when the !D Hand is actually operating, or waiting for the user to press the button (again).
2. One time operations, for example writing an RFID tag or reading detailed information from a single tag. This is an operation where the user presses the button, and then the action will be executed in a short timeframe. The status LED will not blink, and no pause/continue notifications are sent. Sounds will indicate the result of this action.

For now, the !D Hand is only able to work in Europe. In the future, when we'll support other regions, we'll add functionality to retrieve the regions the !D Hand is capable of, and setting the right region. This is currently not implemented in the presentation layer.

iOS

`NSNotificationCenter`⁴ is the generic class for receiving and sending one-to-many events within iOS. Subscribing to events can be done with the `addObserver:` method. For example, to subscribe to the `NHPresentationDidConnectNotification`, use the following code:

```
[[NSNotificationCenter defaultCenter] addObserver:self
    selector:@selector(didConnect:)
    name:@"NHPresentationDidConnectNotification"
    object:nil];
```

The observer is set to the current object, most probably the current view. This code assumes there is a method called `didConnect:` (which is called when the notification fires) and can be implemented like:

```
- (void)didConnect:(NSNotification *)notification
{
    // implement code here...
}
```

Inside each `NSNotification`, there is a `NSDictionary`, containing zero or more key-value pairs that contain information about the notification. For example inside the `NHPresentationDidConnectNotification` there is a key that contains the serial number of the !D Hand. You can obtain these keys according to the following snippet:

```
NSString *serialNumber = [[notification userInfo] valueForKey:@"serialNumber"];
NSLog(@"Serial number: %@", serialNumber)
```

To unsubscribe from a certain event (for example, when switching to a different view), use the `removeObserver:` method. This method removes all notifications that are tied to this specific observer.

```
[[NSNotificationCenter defaultCenter] removeObserver:self];
```

For more examples and in-line explanation, please see the `NedapHandheldAPIDemo` application.

⁴ See "Notification Programming Topics" in the iOS documentation at <http://developer.apple.com/ios>.

Windows

Presentation is a high-level communication layer used to communicate with handheld reader. It is built on top of the session layer and provides more sophisticated functionality. This layer functions asynchronously calling functions in session layer immediately. Once an answer is obtained from the device, presentation layer is notified via event. This layer contains public events and methods for all implemented functionality and it is accessed from GUI application. Use of presentation layer is recommended for connecting to device.

When developing application based up on this API there are few things you have to take care of in order to get full functionality. To properly use API, first you need to create `SerialCommunicationPort`:

```
public SerialCommunicationPort Port = new SerialCommunicationPort(portName, 9600,
    Parity.None, 8, StopBits.One);
```

Port name is name of the `System.IO.Ports.SerialPort` available on the device (example: "COM1"). Port is then used to create `Session`:

```
public Session session = new Session(Port, true);
```

`PresentationFacade` is then created with that `Session`.

```
public PresentationFacade Facade = new PresentationFacade(session);
```

Communication with device is easily done through `PresentationFacade` and all device responses and result data is returned through events. Subscribing to event and calling method via presentation:

```
Program.Controller.Facade.OnReadRfidTag += new System.EventHandler<ReadRfidTagEventArgs>
(facade_OnReadRfidTag);
Program.Controller.Facade.RfidInventoryStart();
```

When updating GUI on triggered event developer must take care of asynchronous GUI update because of the thread security. Example of solving this problem with `BeginInvoke` method:

```
private void facade_OnReadRfidTag(object sender, ReadRfidTagEventArgs e)
{
    lbInventoryItemsList.BeginInvoke(new UpdateUIDelegate(FormUpdateMethod),
        e.EpcObservation);
}

private void FormUpdateMethod(EpcObservation epcObservation)
{
    lbInventoryItemsList.Items.Add(epcObservation.EpcCode.PureIdentityURI);
}
```

Objects

Memory bank

This enumeration defines the memory banks for RFID operations.

enumeration iOS	enumeration Windows	description
NHRFIDMemoryBankReserved	RfidMemoryBank.Reserved	reserved memory bank
NHRFIDMemoryBankEPC	RfidMemoryBank.Epc	EPC memory bank
NHRFIDMemoryBankTID	RfidMemoryBank.Tid	TID memory bank
NHRFIDMemoryBankUserMemory	RfidMemoryBank.UserMemory	user memory bank

Select

The select object can be used to represent a select statement, that can be used for filtering in the air interface protocol. The select object can be set manually with the properties mentioned, or directly with an EPC code (which automatically sets the right properties).

property	type	description
memoryBank	Memory bank	memory bank used in select
addressInBits	Integer	address for start of select
dataLengthInBits	Integer	data length in bits for select
data	Data	actual data
invert	Bool	invert the select statement

Password

The password object can be used to represent any password in the air interface protocol.

property	type	description
hexString	String	password as a hexadecimal string

EPC observation

The EPC observation object is used to represent an actual observation of an EPC tag, also including the Received Signal Strength Indication (RSSI), read count and the timestamp when the tag has been read.

property	type	description
epcCode	EPCCode	EPC code that was read
timestamp	Date	timestamp of the reading
rsi	Integer	received signal strength indication
readCount	Integer	read count in current read cycle - will only be larger than one for Session 0

General

This section describes the general functionality to communicate with the !D Hand. There are also some events that appear without any specific request, i.e. events about the battery status and connection/disconnection. Furthermore, there are also notifications that can appear after any method call.

Methods

Check connection

Check if the handheld is connected.

```
iOS + (BOOL)handheldIsConnected
```

```
Windows bool HandheldIsConnected()
```

Disconnect

Let the handheld disconnect from the device it is connected to.

```
iOS + (void)handheldDisconnect
```

```
Windows void HandheldDisconnect()
```

Battery status

Retrieves the percentage of battery capacity that is remaining in the !D Hand.

```
iOS + (void)handheldGetBatteryPercentage
```

```
Windows void HandheldGetBatteryPercentage()
```

Stop current operation

Stop the current operation, e.g. inventory or monitoring. Can be used when switching from one view to another.

```
iOS + (void)handheldStop
```

```
Windows void HandheldStop()
```

Get information

Get information about the model number of the handheld, the hardware and firmware revision and the serial number.

```
iOS + (NSDictionary *)handheldGetInformation
```

```
Windows DeviceInfo HandheldGetInformation()
```

The data in the dictionary is defined as follows:

parameter	type	description
modelName	String	model number
serialNumber	String	serial number
firmwareRevision	String	firmware revision (x.x.x)
hardwareRevision	String	hardware revision (x.x.x)

Notifications

Connect

The !D Hand connected in the background.

iOS `NHPresentationHandheldDidConnectNotification`

Windows `EventHandler<EventArgs> OnConnect`

key	value type	description
modelName	String	model number
serialNumber	String	serial number
firmwareRevision	String	firmware revision (x.x.x)
hardwareRevision	String	hardware revision (x.x.x)

Please note that the Windows implementation of the API doesn't carry these values for this event. These values should be obtained separately using the 'HandheldGetInformation()' method.

Disconnect

The !D Hand disconnected.

iOS `NHPresentationHandheldDidDisconnectNotification`

Windows `EventHandler<EventArgs> OnDisconnect`

Continue

The user pressed the button to let an action continue (e.g. inventory). This notification can be used to start a 'reading' indication on the screen (e.g. a spinner).

iOS `NHPresentationHandheldDidContinueNotification`

Windows `EventHandler<EventArgs> OnContinue`

Pause

The user pressed the button to let an action pause (e.g. inventory). This notification can be used to stop a 'reading' indication on the screen (e.g. a spinner).

iOS `NHPresentationHandheldDidPauseNotification`

Windows `EventHandler<EventArgs> OnPause`

Low battery

The battery of the !D Hand has only 10% of its capacity remaining. Charge as soon as possible.

iOS `NHPresentationHandheldDidReceiveLowBatteryNotification`

Windows `EventHandler<EventArgs> OnLowBatteryWarning`

Empty battery

The battery is empty now, and the !D Hand will shutdown.

iOS `NHPresentationHandheldDidReceiveEmptyBatteryNotification`

Windows `EventHandler<EventArgs> OnEmptyBatteryStatus`

Battery percentage left

This notification contains the percentage of energy that's left in the battery.

iOS `NHPresentationHandheldDidReceiveBatteryPercentageNotification`

Windows `EventHandler <ReceiveBatteryPercentageEventArgs> OnReceiveBatteryPercentage`

key	value type	description
percentage	Integer	percentage of battery capacity left

User interaction

Enumerations

Tune

Enumeration for the possible tunes the beeper can generate. If you'd need more tunes for specific situations, please let us know.

enumeration iOS	enumeration Windows	description
NHUITuneShort	UITune.Short	short beep (used for 'new tag found')
NHUITuneSuccess	UITune.Success	beep indicating a successful action (used for 'write tag successful')
NHUITuneFailure	UITune.Failure	indicating a failure (used for 'write tag failed')

Methods

Play tune

Plays a tune at the !D Hand. Tunes are defined in the [Tune](#) enumeration.

iOS `+ (void)uiPlayTune:(NHUITune)tune`

Windows `void UIPlay(UITune tune)`

Vibrate

Vibrate the !D Hand.

iOS `+ (void)uiVibrate`

Windows `void UIVibrate()`

Vibrate tune

Vibrates a tune at the !D Hand. Tunes are defined in the [Tune](#) enumeration.

iOS `+ (void)uiVibrateTune:(NHUITune)tune`

Windows void UIVibrateTune(UITune tune)

Notifications

Push button

The user did press the button. This notification is only sent when there is no operation ongoing or waiting for user action.

iOS NHPresentationUIdidPushAButtonNotification

Windows EventHandler <EventArgs> OnPushButton

Shake

The user did shake the device.

iOS NHPresentationUIdidShakeNotification

Windows EventHandler <EventArgs> OnShake

RFID

Enumerations

MDID value

Enumeration for the possible MDID values (chip manufacturer identification).

enumeration iOS	enumeration Windows	description
NHRFIDMDIDImpinj	RfidEpcMdid.Impinj	Impinj
NHRFIDMDIDTexasInstruments	RfidEpcMdid.TexasInstruments	Texas Instruments
NHRFIDMDIDAlienTechnology	RfidEpcMdid.AlienTechnology	Alien Technology
NHRFIDMDIDIntellex	RfidEpcMdid.Intellex	Intellex
NHRFIDMDIDAtmel	RfidEpcMdid.Atmel	Atmel
NHRFIDMDIDNXP	RfidEpcMdid.NXP	NXP
NHRFIDMDIDSTMicroelectronics	RfidEpcMdid.STMicroelectronics	ST Microelectronics
NHRFIDMDIDEPMicroelectronics	RfidEpcMdid.EPMicroelectronics	EP Microelectronics
NHRFIDMDIDMotorola	RfidEpcMdid.Motorola	Motorola
NHRFIDMDIDSentechSndBhd	RfidEpcMdid.SentechSndBhd	Sentech Snd. Bhd.
NHRFIDMDIDEMMicroelectronics	RfidEpcMdid.EMMicroelectronics	EM Microelectronics
NHRFIDMDIDRenesasTechnology	RfidEpcMdid.RenesasTechnology	Renesas Technology
NHRFIDMDIDMstar	RfidEpcMdid.Mstar	Mstar
NHRFIDMDIDTycoInternational	RfidEpcMdid.TycoInternational	Tyco International
NHRFIDMDIDQuanrayElectronics	RfidEpcMdid.QuanrayElectronics	Quanray Electronics
NHRFIDMDIDFujitsu	RfidEpcMdid.Fujitsu	Fujitsu
NHRFIDMDIDLSIS	RfidEpcMdid.LSIS	LSIS
NHRFIDMDIDCaenRFID	RfidEpcMdid.CaenRFID	Caen RFID
NHRFIDMDIDPEG	RfidEpcMdid.PEG	PEG

Methods

Inventory

The inventory methods provide an easy way to do inventory. It automatically filters 'multiple' reads in a buffer and generates a 'beep' on new tags. EPC air interface parameters are set automatically to session 1, target A (no switching) and automatic Q. Register for the notification '[Did read tag](#)' to retrieve read tags.

```
+ (void)rfidInventoryStart
+ (void)rfidInventoryStartWithSelect:(NHSelect *)select
iOS + (void)rfidInventoryStartInBuffer:(NSString *)bufferName
+ (void)rfidInventoryStartInBuffer:(NSString *)bufferName withSelect:(NHSelect *)select
```

```

        void RfidInventoryStart()
        void RfidInventoryStart(Select select)
Windows    void RfidInventoryStart(String bufferName)
        void RfidInventoryStart(String bufferName, Select select)

```

parameter	type	description
select	Select	select object to do filtering of tags in the air interface
bufferName	String	the name of the buffer to store the RFID tags in - multiple buffers can come in handy if there are multiple inventory options in an application, e.g. inventory and shelf availability

Inventory high volume

This method basically does the same as the normal inventory method, but with the exception of using session 2 and 3 for reading; giving better performance in environments with large numbers of tags. When the inventory buffer is cleared, automatically the session will be switched from 2 to 3, and backwards. Please note that if you've read a tag in a session, it will persist its 'read' state (normally) for more than a minute - typically up to 30 minutes - depending on the chip used.

```

        + (void)rfidInventoryHighVolumeStart
        + (void)rfidInventoryHighVolumeStartWithSelect:(NHSelect *)select
iOS    + (void)rfidInventoryHighVolumeStartInBuffer:(NSString *)bufferName
        + (void)rfidInventoryHighVolumeStartInBuffer:(NSString *)bufferName withSelect:(NHSelect *)
        select

        void RfidInventoryHighVolumeStart()
Windows    void RfidInventoryHighVolumeStart(Select select)
        void RfidInventoryHighVolumeStart(String bufferName)
        void RfidInventoryHighVolumeStart(String bufferName, Select select)

```

parameter	type	description
select	Select	select object to do filtering of tags in the air interface
bufferName	String	the name of the buffer to store the RFID tags in - multiple buffers can come in handy if there are multiple inventory options in an application, e.g. inventory and shelf availability

Clear inventory buffer

To clear the inventory buffer, use one of the following methods. After the buffer in the API is cleared, the notification '[Did reset inventory buffer](#)' will fire. If the method without the name of the buffer is used, the current active buffer will be reset - not the 'default' buffer! The default buffer can be erased by using "default" as buffer name.

```

        + (void)rfidInventoryReset
iOS    + (void)rfidInventoryResetBuffer:(NSString *)bufferName

        void RfidInventoryReset()
Windows    void RfidInventoryReset(String bufferName)

```

parameter	type	description
bufferName	String	the name of the buffer to reset

Monitoring

Using the monitoring methods, it's possible to see which tags are in the current field. These methods can be used to implement searching functionality, in combination with the returned RSSI (in the [EPC observation](#) object) and a select

statement. Monitoring uses session 0 in the air interface protocol. For each read cycle, the notification '[Did read tags](#)' will notify the application of the tags read.

```
    + (void)rfidMonitorStart
iOS  + (void)rfidMonitorStartWithSelect:(NHSelect *)select

    void RfidMonitorStart()
Windows void RfidMonitorStart(Select select)
```

parameter	type	description
select	Select	select object to do filtering of tags in the air interface

Single read cycle

This method can be used to do a single read cycle, with low power. The goal of this function is to read the EPC code of a tag that's nearby the handheld, for example to obtain information about a specific product. For the response, see the notification '[Did read tags](#)'.

```
    + (void)rfidReadSingle
iOS  + (void)rfidReadSingleWithSelect:(NHSelect *)select

    void RfidReadSingle()
Windows void RfidReadSingle(Select select)
```

parameter	type	description
select	Select	select object to do filtering of tags in the air interface

Read detailed tag information

To obtain the tag manufacturer ID (MDID), the tag model number, or the tag's extended identifier field (XTID), use this method. The response is provided in a [NHPresentationRFIDDidReadDetailedTagInformationNotification](#).

```
    + (void)rfidReadDetailedTagInformation
iOS  + (void)rfidReadDetailedTagInformationFromTag:(EPCCode *)code

    void RfidReadTag()
Windows void RfidReadTag(EpcCode code)
```

parameter	type	description
code	EPC code	specific tag to read data from (used to create a select statement in the air interface)

Write EPC

Write a new EPC value to a tag, including a verification if there's just one tag in the field, writing to the tag and verifying whether the write action was successful by reading the tag again.

```
iOS  + (void)rfidWriteEPC:(EPCCode *)newEPCCode

Windows void RfidWriteEpc(EpcCode newEpcCode)
```

parameter	type	description
newEPCCode	EPC code	the new EPC value to write to a tag

Notifications

Read tag

A new tag has been found during inventory.

iOS NHPresentationRFIDDidReadTagNotification

Windows EventHandler <ReadRfidTagEventArgs> OnReadRfidTag

key	value type	description
epcObservation	EPC observation	observation object that contains the new tag

Read tags

All tags of one read cycle. This can be used to speed up the processing, to process updates in the screen in batch-wise, instead of tag by tag as with the 'read tag' event.

iOS NHPresentationRFIDDidReadTagsNotification

Windows EventHandler <ReadRfidTagsEventArgs> OnReadRfidTags

key	value type	description
epcObservations	Array	array of EPC Observation

Reset inventory buffer

This notification is sent after the buffer is reset. Use this to synchronize the buffer in your application with the buffer in the API. For a typical example on how to use this, please see the *NedapHandheldAPIDemo* application.

iOS NHPresentationRFIDDidResetInventoryNotification

Windows EventHandler <EventArgs> OnClearInventory

Read detailed tag information

Detailed information about an RFID tag.

iOS NHPresentationRFIDDidReadDetailedTagInformationNotification

Windows EventHandler <ReadDetailedTagInformationEventArgs> OnReadDetailedTagInformation

key	value type	description
tagMDID	Integer	MDID value, corresponding to the MDID value enumeration
tagModelNumber	Integer	model number of the tag
tagXTID	Integer	XTID value

Write of EPC successful

Writing the EPC code did succeed; everything went fine, including verification.

iOS NHPresentationRFIDDidSucceedWriteEPCNotification

Windows EventHandler <EventArgs> OnSucceedWriteEpc

Write of EPC failed: no tags in the field

Writing the EPC code did fail, because there are no tags in the field, or no tags comply to the issued select statement.

iOS NHPresentationRFIDDidFailWriteEPCZeroReadNotification

Windows EventHandler <EventArgs> OnFailWriteEpcZeroRead

Write of EPC failed: more than one tag in the field

Writing the EPC code did fail, because there are more tags than one in the field. This is an error, because otherwise it's unsure which tags should be written. The solution is to move the ID Hand and the tag that needs to be written out of the area of other surrounding tags. A distance of about 20cm should work fine for typical retail tags.

iOS NHPresentationRFIDDidFailWriteEPCMultipleReadNotification

Windows EventHandler <EventArgs> OnFailWriteEpcMultipleRead

Write of EPC failed: write itself failed

Writing the EPC code did fail. This error is returned by the RFID reader, there was an error in the air interface protocol while trying to write to a tag.

iOS NHPresentationRFIDDidFailWriteEPCWriteFailedNotification

Windows EventHandler <EventArgs> OnFailWriteEpcWriteFailed

Write of EPC failed: verification failed

Writing the EPC code did fail, because the verification failed. After a tag has been written successfully, the written value will be verified with an additional read action - to make sure the writing was successful.

iOS NHPresentationRFIDDidFailWriteEPCVerificationFailedNotification

Windows EventHandler <EventArgs> OnFailWriteEpcVerificationFailed

Barcode

Since the barcode reader is an optional component, it's always a good idea to check if it's available, before requesting to read a barcode. Things won't break if you don't, but it's better for the user interaction to do so. On how to do that, please see the example applications.

Enumerations

Barcode type

This enumeration defines the available types of barcodes.

enumeration iOS	enumeration Windows	description
NHBarcodeTypeAll	BarcodeType.All	All supported barcode types
NHBarcodeTypeEAN8	BarcodeType.Ean8	EAN 8
NHBarcodeTypeEAN13	BarcodeType.Ean13	EAN 13
NHBarcodeTypeUPCA	BarcodeType.Upca	UPC-A
NHBarcodeTypeUPCE	BarcodeType.Upce	UPC-E

enumeration iOS	enumeration Windows	description
NHBarcodeTypeEAN128	BarcodeType.Ean128	EAN 128
NHBarcodeTypeCodabar	BarcodeType.Codabar	Codabar
NHBarcodeTypeCode11	BarcodeType.Code11	Code 11
NHBarcodeTypeCode39	BarcodeType.Code39	Code 39
NHBarcodeTypeCode93	BarcodeType.Code93	Code 93
NHBarcodeTypeEANBookland	BarcodeType.Eanbookland	EAN Bookland
NHBarcodeTypeDiscrete2of5	BarcodeType.Discrete2of5	Discrete 2 of 5
NHBarcodeTypeInterleaved2of5	BarcodeType.Interleaved2of5	Interleaved 2 of 5
NHBarcodeTypeCode128	BarcodeType.Code128	Code 128
NHBarcodeTypeMSI	BarcodeType.Msi	MSI
NHBarcodeTypeGS1Databar	BarcodeType.Gs1databar	GS1 Databar
NHBarcodeTypeUnknown	BarcodeType.Unknown	Unknown barcode type

Methods

Barcode reader present check

To check if the barcode reader is present, use the following method.

```

iOS    + (BOOL)barcodeHasReader

Windows  bool BarcodeHasReader()

```

Read barcode

This method reads a barcode. It waits until the user presses the button, then the barcode reader will be enabled for the duration specified in the method call. The duration is specified in seconds, maximum value is 20 seconds. Typical value is 3 seconds. If the 'read types' method is used, the array should consist [Barcode type](#) values (for iOS, inside a NSNumber).

```

iOS    + (void)barcodeReadType:(NHBarcodeType)barcodeType duringSeconds:(NSUInteger)duration
        + (void)barcodeReadTypes:(NSArray *)barcodeTypes duringSeconds:(NSUInteger)duration

Windows  void BarcodeRead(BarcodeType barcodeType, int duration)
        void BarcodeRead(Collection<BarcodeType> barcodeTypes, int duration)

```

Notifications

Read a barcode

Notification received when a barcode has been read.

```

iOS    NHPresentationBarcodeDidReadABarcodeNotification

Windows  EventHandler <ReadBarcodeEventArgs> OnReadBarcode

```


key	value type	description
barcode	String	contains the barcode
type	Barcode type	type of barcode

In iOS, the type is a NSNumber containing an unsigned int with the barcode type.

Not read a barcode

No barcode has been read during the specified duration. If this occurs while you try to scan a real, undamaged barcode, make sure the right type of barcode has been specified in the method call.

iOS NHPresentationBarcodeDidNotReadABarcodeNotification

Windows EventHandler <EventArgs> OnNotReadBarcode

NFC cards

Methods

Read NFC card

This method can be used to try to read NFC cards. It'll enable the field, and continuously try to read cards in the field. By default, it'll only read A-type cards.

iOS + (void)nfcReadStart

Windows void NfcReadStart()

Notifications

Read card

Notification received when a NFC card has been read.

iOS NHPresentationNFCDidReadCardNotification

Windows EventHandler <ReadNfcCardEventArgs> OnReadNfcCard

key	value type	description
cardNumber	Integer	card serial number

6. Session layer

This chapter describes the session layer: a wrapper around the transport layer.

iOS

This layer works asynchronous, like the presentation layer, but with the difference that the responses do not consist of multicast events, but simple callbacks.

Windows

Session layer is a threaded layer that deals with binary protocol communication over Bluetooth serial port, sending commands using communication queue, parsing received data and notifying through events. It is used to communicate with handheld reader and to provide simple function calls for device commands. It contains public methods exposed for usage if needed but it is mainly accessed through presentation layer. This layer uses interfaces for Communication Port, Communication Queue and Connection State.

For connecting and pairing of !D Hand and the phone without user interaction (automatically) BluetoothCommunicationPort class can be used. This class implements ICommunicationPort and uses 32feet.net library that provides support for Microsoft and Broadcom (Widcomm) Bluetooth stacks.

Other option is to use SerialCommunicationPort class which wraps SerialPort class and also implements ICommunicationPort interface. This implementation is Bluetooth stack independent, but requires a user to manually pair and connect phone to !D Hand Bluetooth Serial Port.

Communication Queue defines 2-way threaded communication queue which interacts with communication port and implements ICommunicationPort interface, so that BluetoothCommunicationPort or SerialCommunicationPort class can be used as port implementation.

ConnectionState class is used by the Session for connect/disconnect detection. It implements IConnectionState interface, and runs in separate thread that periodically sends/receives a message.

Management

Methods

Set delegate

Sets the callback delegate for the session layer. The delegate has to conform to the NHSessionDelegateProtocol protocol.

```
iOS + (void)setDelegate:(id <NHSessionDelegateProtocol>)delegate
```

Windows not available yet

parameter	type	description
delegate	id	delegate object for callbacks

General

Methods

Check if connected

Checks if the !D Hand is currently connected. Returns a boolean with the connection status.

```
iOS + (BOOL)handheldIsConnected
```

Windows not available yet

Battery status

Retrieves the percentage that the battery has remaining from the !D Hand.

iOS + (void)handheldGetBatteryPercentage

Windows not available yet

Disconnect

Let the handheld disconnect from the device it is connected to.

iOS + (void)handheldDisconnect

Windows not available yet

Get information

Get information about the model number of the handheld, the hardware and firmware revision and the serial number.

iOS + (NSDictionary *)handheldGetInformation

Windows not available yet

The data in the dictionary is defined as follows:

parameter	type	description
modelName	String	model number
serialNumber	String	serial number
firmwareRevision	String	firmware revision (x.x.x)
hardwareRevision	String	hardware revision (x.x.x)

Callbacks

Connection established

The !D Hand connected in the background.

iOS - (void)sessionHandheldDidConnect:(NSString *)modelName serialNumber:(NSString *)serialNumber firmwareRevision:(NSString *)firmwareRevision hardwareRevision:(NSString *)hardwareRevision

Windows not available yet

parameter	type	description
modelName	String	model number
serialNumber	String	serial number
firmwareRevision	String	firmware revision (x.x.x)
hardwareRevision	String	hardware revision (x.x.x)

Disconnection

The !D Hand disconnected.

iOS - (void)sessionHandheldDidDisconnect

Windows not available yet

Low battery

The battery has only 10% remaining. Charge as soon as possible.

iOS - (void)sessionHandheldDidReceiveLowBatteryWarning

Windows not available yet

Empty battery

The battery is empty now, and the device will shutdown.

iOS - (void)sessionHandheldDidReceiveEmptyBatteryStatus

Windows not available yet

!D Hand started charging

The user started charging the device - no further actions are possible until charging is done.

iOS - (void)sessionHandheldDidReceiveChargingStatus

Windows not available yet

Battery percentage received

This notification contains the percentage of energy that's left in the battery.

iOS - (void)sessionHandheldDidReceiveBatteryPercentage:(NSNumber *)percentage

Windows not available yet

parameter	type	description
percentage	NSNumber	percentage of battery capacity left

Error received

This callback will arise when the user shakes the !D Hand.

iOS - (void)sessionHandheldDidReceiveError:(NSString *)error

Windows not available yet

parameter	type	description
error	String	string containing an error message

User interaction

Methods

Play tune

Plays a tune at the !D Hand. Tunes are defined in the [NHUITune](#) enumeration.

iOS + (void)uiPlayTune:([NHUITune](#))tune

Windows not available yet

Vibrate

Vibrate the !D Hand.

iOS + (void)uiVibrate
+ (void)uiVibrate:(NSInteger)timeout

Windows not available yet

parameter	type	description
timeout	Integer	timeout for vibration in tenths of a second (e.g. 3 equals 0.3 second of vibration)

Vibrate tune

Vibrates a tune at the !D Hand. Tunes are defined in the [NHUITune](#) enumeration.

iOS + (void)uiVibrateTune:([NHUITune](#))tune

Windows not available yet

Status LED

Set the status LED to blinking or not.

iOS + (void)uiSetStatusLedBlinking:(BOOL)blinking

Windows not available yet

parameter	type	description
blinking	BOOL	sets the blinking: true is enabled, false is disabled

Callbacks

Button

This callback will arise when the user presses the button.

iOS - (void)sessionUIDidPushButton

Windows not available yet

Shake

This callback will arise when the user shakes the !D Hand.

iOS - (void)sessionUIDidShake

Windows - not available yet

RFID

Enumerations

NHRFIDRegion

This enumeration defines the regions for RFID operations⁵.

enumeration	description
NHRFIDRegionUndefined	Undefined region
NHRFIDRegionEurope	Europe, according to ETSI EN 302 208
NHRFIDRegionNorthAmerica	North America, according to FCC (not supported at the moment)
NHRFIDRegionKorea	Korea (not supported at the moment)
NHRFIDRegionChina	China (not supported at the moment)

NHRFIDFrequency

This enumeration defines the frequencies that can be used to set the frequency hop table.

enumeration	description
NHRFIDFrequencyEurope865700	865.7 MHz
NHRFIDFrequencyEurope866300	866.3 MHz
NHRFIDFrequencyEurope866900	866.9 MHz
NHRFIDFrequencyEurope867500	867.5 MHz

NHRFIDSession

This enumeration defines the session to be used in the EPC air interface protocol.

enumeration	description
NHRFIDSession0	Session 0 (no persistence)
NHRFIDSession1	Session 1 (persistence of about 1 second)
NHRFIDSession2	Session 2 (persistence of more than 60 seconds)
NHRFIDSession3	Session 3 (persistence of more than 60 seconds)

⁵ The reason these regions are not supported at the moment, is because of the antenna design. The included reader supports all of the mentioned regions. Over time we'll support additional frequency regions by means of an adopted antenna as well.

NHRFIDTarget

This enumeration defines the target to be used in the EPC air interface protocol.

enumeration	description
NHRFIDTargetA	target A
NHRFIDTargetB	target B

NHRFIDMValue

This enumeration defines the M-value to be used in the EPC air interface protocol.

enumeration	description
NHRFIDMValue0	M-value of 0
NHRFIDMValue2	M-value of 2
NHRFIDMValue4	M-value of 4
NHRFIDMValue8	M-value of 8

Methods

Get supported regions

Asks the !D Hand which regions are supported.

iOS + (void)rfidGetSupportedRegions

Windows not available yet

Set region

Sets the region for RFID reading.

iOS + (void)rfidSetRegion:([NHRFIDRegion](#))region

Windows not available yet

parameter	type	description
region	NHRFIDRegion	region to be used for rfid operations

Set frequency hop table

By default, all channels in a region are used for hopping. Using this method it's possible to set a specific hop table, thus use only the frequencies that you want to use.

iOS + (void)rfidSetFrequencyHopTable:(NSArray *)frequencies

Windows not available yet

parameter	type	description
frequencies	NSArray	array of frequencies (NSNumber containing NHRFIDFrequency)

Set session

Sets the session for the air interface protocol.

iOS + (void)rfidSetSession:([NHRFIDSession](#))session

Windows not available yet

parameter	type	description
session	NHRFIDSession	the session to use

Set target

Sets the target for the air interface protocol.

iOS + (void)rfidSetTarget:([NHRFIDTarget](#))target toggle:(BOOL)toggle

Windows not available yet

parameter	type	description
target	NHRFIDTarget	the target to use
toggle	BOOL	switch between target A and B

Set M-value

Sets the M-value for the air interface protocol.

iOS + (void)rfidSetMValue:([NHRFIDMValue](#))mValue

Windows not available yet

parameter	type	description
mValue	NHRFIDMValue	the M-value to use

Set Q-value

Sets the Q-value for the air interface protocol.

iOS + (void)rfidSetQValue:(NSInteger)qValue dynamic:(BOOL)dynamic

Windows not available yet

parameter	type	description
qValue	Integer	the Q-value to use (between 0 and 16)
dynamic	BOOL	whether the Q-value is dynamically adjusted

Read EPC

Read EPC tags in the field for a specified duration and with a specified power.

iOS + (void)rfidReadEPCsWithTimeout:(NSInteger)timeout power:(NSInteger)power select:(NHSelect *)select

Windows not available yet

parameter	type	description
timeout	Integer	timeout in milliseconds
power	Integer	the power, possible values range from 10 to 23 dBm
select	Select	select statement, can also be set to nil

Read data

Read data from an EPC tag in the field.

iOS + (void)rfidReadData:(NHRFIDMemoryBank)memoryBank addressInWords:(NSInteger)addressInWords lengthInWords:(NSInteger)lengthInWords timeout:(NSInteger)timeout power:(NSInteger)power select:(NHSelect *)select password:(NHPassword *)password

Windows not available yet

parameter	type	description
memoryBank	NHRFIDMemoryBank	the memory bank to read from
addressInWords	Integer	start address in words
lengthInWords	Select	length of to be read data in words
timeout	Integer	timeout in milliseconds
power	Integer	the power, possible values range from 10 to 23 dBm
select	Select	select statement, can also be set to nil
password	NHPassword	optional password, can also be set to nil

Write EPC

Write a new EPC value to a tag. The handheld first verifies if there's just one tag in the field. It then writes the new EPC to the tag (and automatically corrects the PC word if necessary). Finally, it verifies what's written.

iOS + (void)rfidWriteEPC:(EPCCode *)epc;

Windows not available yet

parameter	type	description
epc	EPCCode	the epc to be written

Write data

Write data to an EPC tag in the field.

iOS + (void)rfidWriteData:(NSData *)someData memoryBank:(NHRFIDEPCMemoryBank)memoryBank addressInWords:(NSInteger)addressInWords timeout:(NSInteger)timeout power:(NSInteger)power select:(NHSelect *)select password:(NHPassword *)password;

Windows not available yet

parameter	type	description
someData	Data	the data to be written
memoryBank	NHRFIDEPCMemoryBank	the memory bank to write to
addressInWords	Integer	start address in words
timeout	Integer	timeout in milliseconds
power	Integer	the power, possible values range from 10 to 23 dBm
select	Select	select statement, can also be set to nil
password	NHPassword	optional password, can also be set to nil

Callbacks

Receive supported regions

The list of supported regions is received.

iOS - (void)sessionRFIDDidReceiveSupportedRegions:([NHRFIDRegion](#))regionFlags

Windows not available yet

parameter	type	description
regionFlags	NHRFIDRegion	flags of all supported regions

Tag read

A RFID tag has been read.

iOS - (void)sessionRFIDDidReadTag:([NHEPCObservation](#) *)epcObservation

Windows not available yet

parameter	type	description
epcObservation	NHEPCObservation	observation of the read EPC code of the tag

Read cycle finished

The current read cycle is finished and all read RFID tags are downloaded.

iOS - (void)sessionRFIDDidFinishReadCycle

Windows not available yet

Data read

Data has been read from a RFID tag. An indication is included to indicate whether there are more tags in the field.

iOS - (void)sessionRFIDDidReadData:(NSData *)data moreTagsAvailable:(BOOL)available

Windows not available yet

parameter	type	description
data	Data	the data that has been read from the tag
available	BOOL	indication to indicate whether more tags were available in the field than the one that data was read from

Data written

Data has been written to a RFID tag. An indication is included to indicate whether there are more tags in the field.

iOS - (void)sessionRFIDDidWriteDataMoreTagsAvailable:(BOOL)available

Windows not available yet

parameter	type	description
available	BOOL	indication to indicate whether more tags were available in the field than the one that data was written to

Action failed

Indication that a specific action (read data, write, lock, kill) to a RFID tag has failed. An indication is included to indicate whether there are more tags in the field.

iOS - (void)sessionRFIDDidFailButTagsAvailable:(BOOL)available

Windows not available yet

parameter	type	description
available	BOOL	indication to indicate whether more tags were available in the field than the one that was tried the action on

Barcode

Methods

Barcode reader present

To check if the barcode reader is present, use the following method.

iOS + (BOOL)barcodeHasReader

Windows not available yet

Set barcode type

Set whether a specific barcode type is enabled or not for scanning.

iOS + (void)barcodeSetStatusForType:([NHBarcodeType](#))barcodeType enabled:(BOOL)enabled

Windows not available yet

parameter	type	description
barcodeType	NHBarcodeType	type of barcode
enabled	BOOL	enabled or disabled

Read barcode

Try to read a barcode for a specified duration.

iOS + (void)barcodeTryDecode:(NSUInteger)duration

Windows not available yet

parameter	type	description
timeout	Integer	duration in seconds

Callbacks

Barcode read

A barcode has been read.

iOS - (void)sessionBarcodeDidReadABarcode:(NSString *)barcode barcodeType:(NHBarcodeType)type

Windows not available yet

parameter	type	description
barcode	String	the barcode that was read
type	NHBarcodeType	type of barcode that was read

Barcode not read

A barcode has not been read during the specified duration.

iOS - (void)sessionBarcodeDidNotReadABarcode

Windows not available yet

NFC

Methods

Try to read card

Try to read a card using this method.

iOS + (void)nfcTryToReadCard

Windows not available yet

Callbacks

NFC card read

A NFC card has been read.

iOS - (void)sessionNFCDidReadCard:(NSUInteger)cardNumber

Windows - not available yet

parameter	type	description
cardNumber	Integer	card serial number of the NFC card that was read

NFC card not found

A NFC card has not been found.

iOS - (void)sessionNFCDidNotFoundACard

Windows - not available yet

7. Transport layer

For communication with the !D Hand there is a protocol used. For this protocol there is a message use as described below. If you use the presentation layer you don't need to use this messages. A message to the !D Hand needs to be implemented as shown in the example. In this manual you can find the descriptions of the Data ID's used in the communication.

The communication between the device and the !D Hand is asynchronous.

Packet structure

There are two valid packet structures. Those are described in the next sections.

'Big' packet structure (0x07)

This new packet structure was defined to support larger payload lengths (longer than 256 bytes), and does replace the 'old' packet structure from firmware 1.1 of the !D Hand. This packet structure is only valid for **responses from the !D Hand**. !D Hand's with older firmware (< 1.1) still use the 'small' packet structure.

byte number	value	description
0	0x07	start of packet
1	0xNN	length of packet payload (MSB)
2	0xNN	length of packet payload (LSB)
3	0xNN	command ID
4	0xNN	data bytes (optional)
last byte	0xNN	checksum

'Small' packet structure (0x06)

The 'small' packet structure is still the only valid packet structure for **commands to the !D Hand**. It remains to be supported for **responses from the !D Hand** as well, to support !D Hand's with firmware < 1.1.

byte number	value	description
0	0x06	start of packet
1	0xNN	length of packet payload
2	0xNN	command ID
3	0xNN	data bytes (optional)
last byte	0xNN	checksum

The length of the payload is defined as the sum of the data length plus three (bytes for length, command ID and CRC). So, if there are no data bytes, the length is three. For one data byte, the length is four, etc.

$$\text{length} = \text{length}(\text{data}) + 3$$

To calculate the checksum, the following steps need to be taken:

1. Take the sum of the length byte, the command ID and all the data bytes.
2. Of the sum, take the two least-significant bytes. (`uint8_t lsb = sum & 0xFF`)
3. The checksum is the difference between 0x100 and this value.

For examples, see the command description later on.

Commands to !D Hand

This section describes the commands from an external device, to the !D Hand.

General

0x01 - get battery percentage

Obtain the remaining battery capacity of the !D Hand.

byte number	value	description
0	0x06	start of packet
1	0x03	length of packet payload
2	0x01	command ID
3	0xFC	checksum

0x02 - get device details

Obtain information from the !D Hand (serial number, etc.).

byte number	value	description
0	0x06	start of packet
1	0x03	length of packet payload
2	0x02	command ID
3	0xFB	checksum

0x03 - disconnect

Disconnect the Bluetooth connection.

byte number	value	description
0	0x06	start of packet
1	0x03	length of packet payload
2	0x03	command ID
3	0xFA	checksum

User interaction

0x41 - set status LED

Set the status LED.

byte number	value	description
0	0x06	start of packet
1	0x04	length of packet payload

byte number	value	description
2	0x41	command ID
3	0xNN	turn off (0x00), turn on (0x01) or start blinking (0x02)
last byte	0xNN	checksum

0x42 - play tune

Play a tune on the beeper of the !D Hand.

byte number	value	description
0	0x06	start of packet
1	0x04	length of packet payload
2	0x42	command ID
3	0xNN	tune
last byte	0xNN	checksum

Available tunes

value	description
0x00	short beep (used for 'new tag found')
0x01	beep indicating a successful action (used for 'write tag successful')
0x02	indicating a failure (used for 'write tag failed')

0x43 - vibrate

Vibrate the !D Hand. There are two options; vibration with a fixed length or vibration with a defined length. In the first option, there is no data byte, for the latter option there is one data byte.

Without duration

byte number	value	description
0	0x06	start of packet
1	0x03	length of packet payload
2	0x43	command ID
3	0xNN	checksum

With duration

byte number	value	description
0	0x06	start of packet
1	0x04	length of packet payload
2	0x43	command ID
3	0xNN	duration in tenth of a second (e.g. 3 means 0.3 seconds)

byte number	value	description
4	0xNN	checksum

0x44 - vibrate tune

Play a tune on the vibration motor of the !D Hand.

byte number	value	description
0	0x06	start of packet
1	0x04	length of packet payload
2	0x44	command ID
3	0xNN	tune (see available tunes in command 0x42)
last byte	0xNN	checksum

RFID

0x23 - read EPC

Reads EPC tags for a specified duration.

byte number	value	description
0	0x06	start of packet
1	0xNN	length of packet payload
2	0x23	command ID
3	0xNN	flags
NN	0xNN	select statements (optional, only if indicated in flags)
NN+1	0xNN	timeout in milliseconds (MSB)
NN+2	0xNN	timeout in milliseconds (LSB)
NN+3	0xNN	power in dBm (minimum: 10, maximum: 23)
NN+4	0xNN	checksum

Flags

value	description
0x01	packet contains select
0x02	select inverted
0x10	packet contains password (not valid for EPC reading, command ID 0x23)

Select statement

byte number	value	description
0	0xNN	length of select statement (excluding this byte)

byte number	value	description
1	0xNN	memory bank
2	0xNN	start address in bits (MSB)
3	0xNN	start address in bits (LSB)
4	0xNN	data length in bits (MSB)
5	0xNN	data length in bits (LSB)
NN	0xNN	select data bytes

Password

byte number	value	description
0	0xNN	password (MSB)
1	0xNN	password
2	0xNN	password
3	0xNN	password (LSB)

Example

byte number	value	description
0	0x06	start of packet
1	0x07	length of packet payload
2	0x23	command ID
3	0x00	flags: no select, no password
4	0x01	timeout in milliseconds (MSB): 400ms
5	0x90	timeout in milliseconds (LSB)
6	0x17	power in dBm: 23 dBm
7	0x2E	checksum: (0x07+0x23+0x00+0x01+0x90+0x17+0x2E) & 0xFF = 0x00

0x24 - read data

Read data from a RFID tag.

byte number	value	description
0	0x06	start of packet
1	0xNN	length of packet payload
2	0x24	command ID
3	0xNN	flags
NN	0xNN	select statements (optional, only if indicated in flags)

byte number	value	description
NN+1	0xNN	password (optional, only if indicated in flags, MSB)
NN+2	0xNN	password (optional, only if indicated in flags)
NN+3	0xNN	password (optional, only if indicated in flags)
NN+4	0xNN	password (optional, only if indicated in flags, LSB)
NN+5	0xNN	timeout in milliseconds (MSB)
NN+6	0xNN	timeout in milliseconds (LSB)
NN+7	0xNN	power in dBm (minimum: 10, maximum: 23)
NN+8	0xNN	memory bank to read data from
NN+9	0xNN	start address in words (MSB)
NN+10	0xNN	start address in words (LSB)
NN+11	0xNN	length in words
NN+12	0xNN	checksum

Memory bank

value	description
0x00	reserved memory bank
0x01	EPC memory bank
0x02	TID memory bank
0x03	user memory bank

Example

byte number	value	description
0	0x06	start of packet
1	0x0B	length of packet payload
2	0x24	command ID
3	0x00	flags: no select, no password
4	0x00	timeout in milliseconds (MSB): 100ms
5	0x64	timeout in milliseconds (LSB)
6	0x0A	power in dBm: 10 dBm
7	0x02	memory bank to read data from: TID memory
8	0x00	start address in words (MSB): 0
9	0x00	start address in words (LSB)
10	0x02	length in words: 2

byte number	value	description
11	0x5F	checksum: (0x0B+0x24+0x00+0x00+0x64+0x0A+0x02+0x00+0x00+0x02+0x5F) & 0xFF = 0x00

0x25 - write data

Write data to a RFID tag.

byte number	value	description
0	0x06	start of packet
1	0xNN	length of packet payload
2	0x25	command ID
3	0xNN	flags
NN	0xNN	select statements (optional, only if indicated in flags)
NN+1	0xNN	password (optional, only if indicated in flags, MSB)
NN+2	0xNN	password (optional, only if indicated in flags)
NN+3	0xNN	password (optional, only if indicated in flags)
NN+4	0xNN	password (optional, only if indicated in flags, LSB)
NN+5	0xNN	timeout in milliseconds (MSB)
NN+6	0xNN	timeout in milliseconds (LSB)
NN+7	0xNN	power in dBm (minimum: 10, maximum: 23)
NN+8	0xNN	memory bank to write to
NN+9	0xNN	start address in words (MSB)
NN+10	0xNN	start address in words (LSB)
NN+11	0xNN	data length in words
NN+12	0xMM	data
NN+MM+1	0xMM	checksum

0x26 - lock tag

Lock a RFID tag.

byte number	value	description
0	0x06	start of packet
1	0xNN	length of packet payload
2	0x26	command ID
3	0xNN	flags
NN	0xNN	select statements (optional, only if indicated in flags)
NN+1	0xNN	password (optional, only if indicated in flags, MSB)

byte number	value	description
NN+2	0xNN	password (optional, only if indicated in flags)
NN+3	0xNN	password (optional, only if indicated in flags)
NN+4	0xNN	password (optional, only if indicated in flags, LSB)
NN+5	0xNN	timeout in milliseconds (MSB)
NN+6	0xNN	timeout in milliseconds (LSB)
NN+7	0xNN	power in dBm (minimum: 10, maximum: 23)
NN+8	0xNN	mask bits (MSB)
NN+9	0xNN	mask bits (LSB)
NN+10	0xNN	lock bits (MSB)
NN+11	0xNN	lock bits (LSB)
NN+12	0xMM	checksum

Lock bits, mask bits

value	description
0x0000	lock user memory permanent
0x0001	lock user memory read/write
0x0002	lock TID memory permanent
0x0004	lock TID memory read/write
0x0010	lock EPC memory permanent
0x0020	lock EPC memory read/write
0x0040	lock access password permanent
0x0080	lock access password read/write
0x0100	lock kill password permanent
0x0200	lock kill password read/write

0x27 - kill tag

Kill a RFID tag.

byte number	value	description
0	0x06	start of packet
1	0xNN	length of packet payload
2	0x27	command ID
3	0xNN	flags
NN	0xNN	select statements (optional, only if indicated in flags)

byte number	value	description
NN+1	0xNN	password (optional, only if indicated in flags, MSB)
NN+2	0xNN	password (optional, only if indicated in flags)
NN+3	0xNN	password (optional, only if indicated in flags)
NN+4	0xNN	password (optional, only if indicated in flags, LSB)
NN+5	0xNN	timeout in milliseconds (MSB)
NN+6	0xNN	timeout in milliseconds (LSB)
NN+7	0xNN	power in dBm (minimum: 10, maximum: 23)
NN+8	0xNN	checksum

0x28 - set region

Sets the region for RFID reading.

byte number	value	description
0	0x06	start of packet
1	0x04	length of packet payload
2	0x28	command ID
3	0xNN	region
4	0xNN	checksum

Region

value	description
0x00	undefined
0x01	Europe
0x02	North America
0x04	Korea
0x08	China

0x29 - set EPC parameter

Set one of the EPC parameters for reading.

byte number	value	description
0	0x06	start of packet
1	0xNN	length of packet payload
2	0x29	command ID
3	0xNN	EPC parameter key
4	0xNN	EPC parameter value first

byte number	value	description
5	0xNN	EPC parameter value second (only for target, Q value)
6	0xNN	checksum

EPC parameter key

value	description
0x00	session
0x01	target
0x02	M value
0x03	Q value

Session

value	description
0x00	session 0
0x01	session 1
0x02	session 2
0x04	session 3

Target - first value

value	description
0x00	target A
0x01	target B

Target - second value

value	description
0x00	non toggling
0x01	toggling

M value

value	description
0x00	0
0x01	2
0x02	4
0x03	8

Q value - first value

value	description
0x00	non dynamic

value	description
0x01	dynamic (first value is irrelevant)

Q value - second value

value	description
0x00	0
0x01	1
0x02	2
...	...

0x2A - set CW signal

Enable or disable continuous wave signal.

byte number	value	description
0	0x06	start of packet
1	0x05	length of packet payload
2	0x2A	command ID
3	0xNN	enable CW (0x01) or not (0x00)
4	0xNN	use antenna one (0x00) or antenna two (0x01)
5	0xNN	checksum

0x2B - set frequency hop table

Sets the frequency hop table - a minimum of one frequency should be set, using four bytes. The length of the data bytes should be a multiple of four. Please make sure that the frequencies that are set are compliant with the region for RFID operations.

byte number	value	description
0	0x06	start of packet
1	0xNN	length of packet payload
2	0x2B	command ID
3	0xNN	frequency 1 (MSB)
4	0xNN	frequency 1
5	0xNN	frequency 1
6	0xNN	frequency 1 (LSB)
NN	0xNN	frequency 2 (optional, MSB)
NN	0xNN	frequency 2 (optional)
NN	0xNN	frequency 2 (optional)
NN	0xNN	frequency 2 (optional, LSB)

byte number	value	description
NN+1	0xNN	checksum

0x2C - get supported regions

Retrieve all the supported regions from the !D Hand.

byte number	value	description
0	0x06	start of packet
1	0x03	length of packet payload
2	0x2C	command ID
3	0xD1	checksum

0x2D - write EPC

Write a new EPC value to a tag. The handheld first verifies if there's just one tag in the field. It then writes the new EPC to the tag (and automatically corrects the PC word if necessary). Finally, it verifies what's written.

byte number	value	description
0	0x06	start of packet
1	0xNN	length of packet payload
2	0x2D	command ID
3	0x00	RFU (reserved for future use, set to 0x00)
4	0x00	RFU (reserved for future use, set to 0x00)
5	0xNN	new EPC
NN+1	0xNN	checksum

Barcode

0x31 - set barcode type

Enable or disable a certain barcode type.

byte number	value	description
0	0x06	start of packet
1	0x05	length of packet payload
2	0x31	command ID
3	0xNN	barcode type
4	0xNN	enable (0x01) or disable (0x00)
5	0xNN	checksum

Barcode type

value	description
0x00	all supported barcode types
0x01	EAN 8
0x02	EAN 13
0x03	UPC-A
0x04	UPC-E
0x05	EAN 128
0x06	Codabar
0x07	Code 11
0x08	Code 39
0x09	Code 93
0x0A	EAN Bookland
0x0B	Discrete 2 of 5
0x0C	Interleaved 2 of 5
0x0D	Code 128
0x0E	MSI
0x0F	GS1 Databar
0xFF	unknown barcode type

0x32 - try to read a barcode

Enable the laser and try to read a barcode for the specified duration

byte number	value	description
0	0x06	start of packet
1	0x04	length of packet payload
2	0x32	command ID
3	0xNN	duration in tenth of a second (e.g. 3 means 0.3 seconds)
4	0xNN	checksum

NFC

0x51 - try to read card

Tries to read an NFC card.

byte number	value	description
0	0x06	start of packet
1	0x03	length of packet payload

byte number	value	description
2	0x51	command ID
3	0xAC	checksum

Responses from !D Hand

Errors

0xFF - error

An error has occurred.

byte number	value	description
0	0x06	start of packet
1	0x04	length of packet payload
2	0xFF	command ID
3	0xNN	error code
4	0xNN	checksum

Error code

value	description
0x00	CRC error in command sent to !D Hand
0x01	wrong number of bytes in command sent to !D Hand
0x21	rfid: analog frontend not enabled - error should not occur
0x22	rfid: temperature exceed limits - error should not occur
0x23	rfid: high return loss in antenna - hardware error, return for service
0x24	rfid: busy - error should not occur
0x25	rfid: invalid opcode - error should not occur
0x31	barcode: not available - can occur if no barcode reader is present, and a barcode command is sent

General

0x81 - battery percentage

Percentage of battery capacity remaining.

byte number	value	description
0	0x06	start of packet
1	0x04	length of packet payload
2	0x81	command ID
3	0xNN	percentage remaining (between 0 and 100)

byte number	value	description
4	0xNN	checksum

0x82 - device details

Details about the !D Hand. Serial number, firmware and hardware versions. The firmware and hardware version are defined as, for example version 2.1.3, where:

- 2 is first digit,
- 1 is middle digit,
- 3 is last digit.

byte number	value	description
0	0x06	start of packet
1	0x13	length of packet payload
2	0x82	command ID
3	0xNN	Nedap production code: year
4	0xNN	Nedap production code: month
5	0xNN	Nedap production code: day - left char
6	0xNN	Nedap production code: day - right char
7	0xNN	Nedap production code: modification char
8	0xNN	Nedap production code: serial - left char
9	0xNN	Nedap production code: serial
10	0xNN	Nedap production code: serial - right char
11	0xNN	SKU (MSB)
12	0xNN	SKU
13	0xNN	SKU
14	0xNN	SKU (LSB)
15	0xNN	firmware version (first digit)
16	0xNN	firmware version (middle digit)
17	0xNN	firmware version (last digit)
18	0xNN	hardware version (first digit)
19	0xNN	hardware version (middle digit)
20	0xNN	hardware version (last digit)
21	0xNN	checksum

0x83 - low battery

The battery has just 10% of its capacity left.

byte number	value	description
0	0x06	start of packet
1	0x03	length of packet payload
2	0x83	command ID
3	0x7A	checksum

0x84 - empty battery

The battery is empty: the !D Hand will shutdown now.

byte number	value	description
0	0x06	start of packet
1	0x03	length of packet payload
2	0x84	command ID
3	0x79	checksum

0x85 - start charging

A charger is connected and the !D Hand will start charging.

byte number	value	description
0	0x06	start of packet
1	0x03	length of packet payload
2	0x85	command ID
3	0x78	checksum

User interaction

0xC1 - button pushed

The user pushed the button.

byte number	value	description
0	0x06	start of packet
1	0x03	length of packet payload
2	0xC1	command ID
3	0x3C	checksum

0xC2 - shake

The user shook the !D Hand.

byte number	value	description
0	0x06	start of packet

byte number	value	description
1	0x03	length of packet payload
2	0xC2	command ID
3	0x3B	checksum

RFID

0xA1 - no tags read

Sent only when no tags were read during a read cycle.

byte number	value	description
0	0x06	start of packet
1	0x04	length of packet payload
2	0xA1	command ID
3	0x00	no tags read
4	0x5B	checksum

0xA2 - tags

Tags that were read during a read cycle. There can be multiple of these packets right behind each other. The last packet flag indicates whether one is the last of the current read cycle or not.

byte number	value	description
0	0x06	start of packet
1	0xNN	length of packet payload
2	0xA2	command ID
3	0xNN	last packet (0x01) or more packets are coming (0x00)
4	0xNN	number of tags in this packet
5	0xNN	tag 1: read count
6	0xNN	tag 1: RSSI
7	0xNN	tag 1: EPC length in bits (MSB)
8	0xNN	tag 1: EPC length in bits (LSB)
9	0xNN	tag 1: PC word (MSB)
10	0xNN	tag 1: PC word (LSB)
NN	0xNN	tag 1: EPC data (MSB), length is according to the EPC length
NN	0xNN	...
NN	0xNN	tag 1: EPC data (LSB)
NN+1	0xNN	tag 1: CRC (MSB)

byte number	value	description
NN+2	0xNN	tag 1: CRC (LSB)
NN+3	0xNN	tag 2: ...
NN+MM+1	0xNN	checksum

0xA3 - read data

Sent when it was tried to read data from a tag. The tag data bytes are only present if the action was a success.

byte number	value	description
0	0x06	start of packet
1	0xNN	length of packet payload
2	0xA3	command ID
3	0xNN	number of tags in the field
4	0xNN	action was a success (0x01) or not (0x00)
NN	0xNN	tag data (MSB)
NN	0xNN	...
NN	0xNN	tag data (LSB)
NN	0xNN	checksum

0xA4 - write data

Sent when it was tried to write data to a tag.

byte number	value	description
0	0x06	start of packet
1	0x05	length of packet payload
2	0xA4	command ID
3	0xNN	number of tags in the field
4	0xNN	action was a success (0x01) or not (0x00)
5	0xNN	checksum

0xA5 - supported regions

The regions the !D Hand supports.

byte number	value	description
0	0x06	start of packet
1	0x04	length of packet payload
2	0xA5	command ID
3	0xNN	region flags that are supported

byte number	value	description
4	0xNN	checksum

0xA6 - write EPC

Send when it was tried to write a new EPC to a tag

byte number	value	description
0	0x06	start of packet
1	0x04	length of packet payload
2	0xA6	command ID
3	0xNN	status
4	0xNN	checksum

Status

value	description
0x00	write failed: no tags in the field
0x01	write failed: too much tags in the field (more than one)
0x02	write failed: other reason
0x03	write successful

Barcode

0xB1 - barcode read

A barcode was read. The length of the barcode can be evaluated by looking at the packet payload length. The barcode is encoded as ASCII characters.

byte number	value	description
0	0x06	start of packet
1	0xNN	length of packet payload
2	0xB1	command ID
3	0xNN	barcode type
NN	0xNN	barcode contents (MSB)
NN	0xNN	...
NN	0xNN	barcode contents (LSB)
NN+1	0xNN	checksum

0xB2 - no barcode read

No barcode was read during the specified period.

byte number	value	description
0	0x06	start of packet
1	0x03	length of packet payload
2	0xB2	command ID
3	0x4B	checksum

NFC

0xD1 - no card found

No card was found while trying to read.

byte number	value	description
0	0x06	start of packet
1	0x03	length of packet payload
2	0xD1	command ID
3	0x2C	checksum

0xD2 - card found

A card was found while trying to read.

byte number	value	description
0	0x06	start of packet
1	0x07	length of packet payload
2	0xD2	command ID
3	0xNN	card serial number (MSB)
4	0xNN	card serial number
5	0xNN	card serial number
6	0xNN	card serial number (LSB)
7	0xNN	checksum

8. Getting started with development

This chapter will present the steps that need to be done to equip an empty project with the necessary libraries and links to get started on !D Hand application development.

iOS (in Xcode 4.0)

Make sure to target iOS 4.0 as minimum, due to use of functions in the API only available in iOS > 4.0. Please note that working with external accessories is not supported from within the simulator.

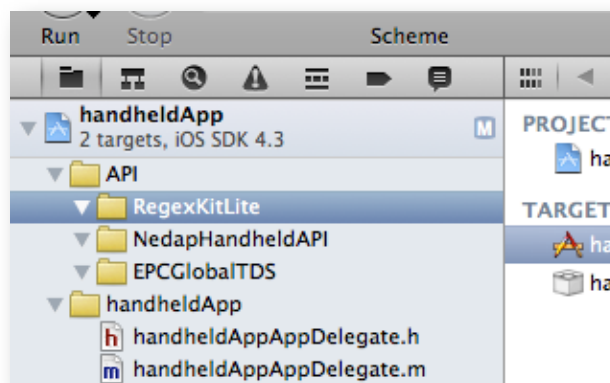
Create a new project

Create a new project inside Xcode; the exact type of project doesn't matter.



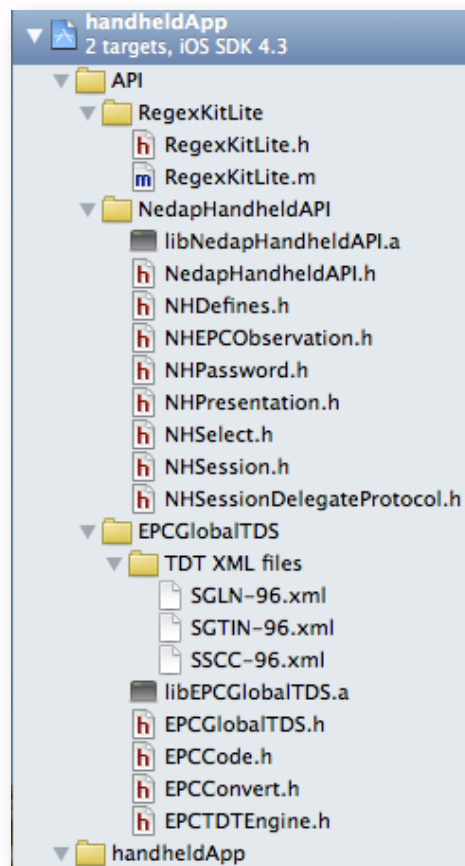
Add API files

Create new groups inside your project (EPCGlobalTDS, NedapHandheldAPI, RegexKitLite) for our API and 3rd party libraries/files. The RegexKitLite library⁶ is used for regular expressions inside the EPCGlobalTDS library.

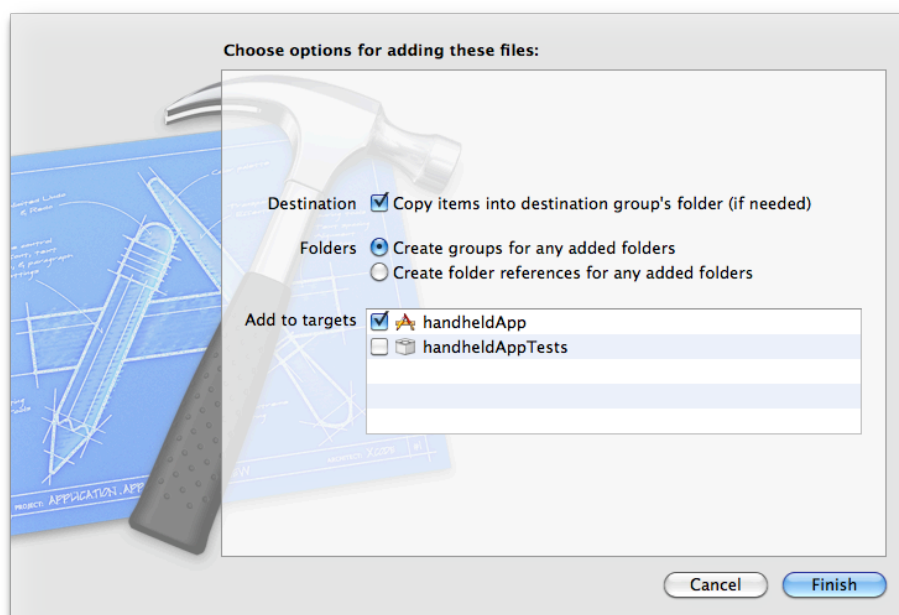


⁶ <http://regexkit.sourceforge.net/RegexKitLite/>

Drag and drop all files (like below) in the project. For the library binaries (.a) files - use the release versions. Also include the relevant Tag Data Translation XML files.

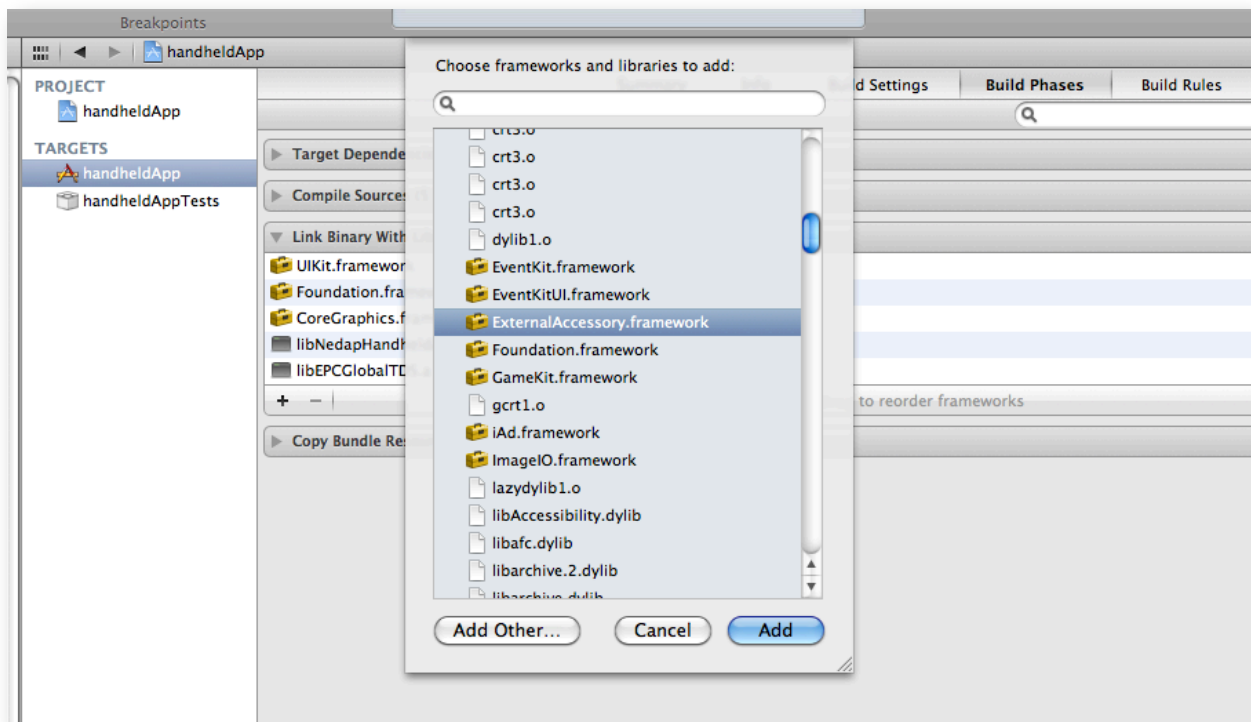


Select 'Copy items into destination group's folder (if needed)' to make sure the files are copied to the project folder.



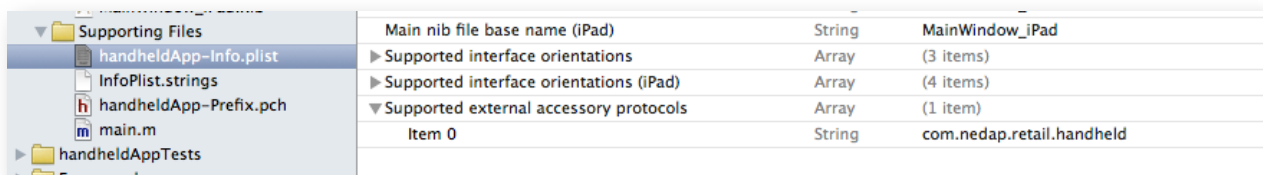
Add framework

Navigate to the project settings, select the right target and in 'Build Settings', add the 'ExternalAccessory.framework'.



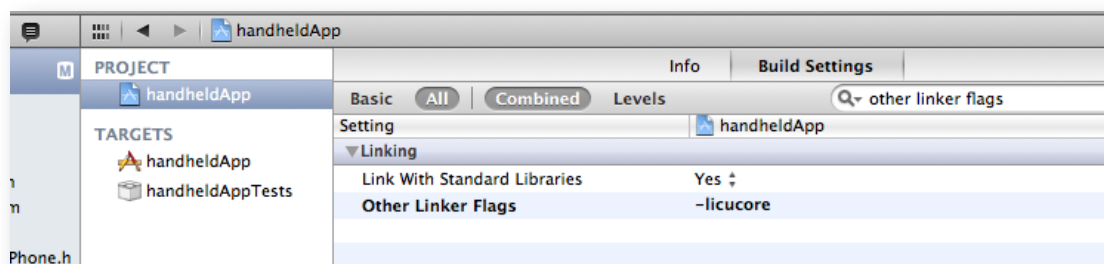
Change settings in .plist file

To make sure that the app is recognized as cooperating with the !D Hand, it needs to be added as a 'supported external accessory protocol' in the .plist file of the project. The name of the protocol is 'com.nedap.retail.handheld'. Please see the screenshot for more details.



Add linker flag

For the RegexKitLite library, it is necessary to add a linker flag. Go to the project Build Settings, select 'All', and look for 'Other Linker Flags'. Enter here '-licucore'.



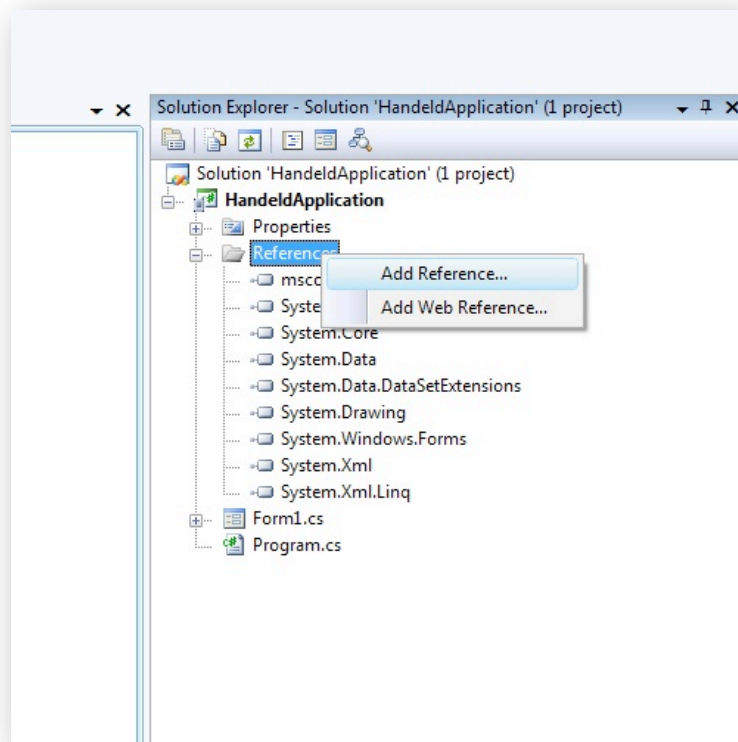
Attribute Open-Source project 'RegexKitLite' in your application [1] - see their website for more information. It uses a permissive BSD license.

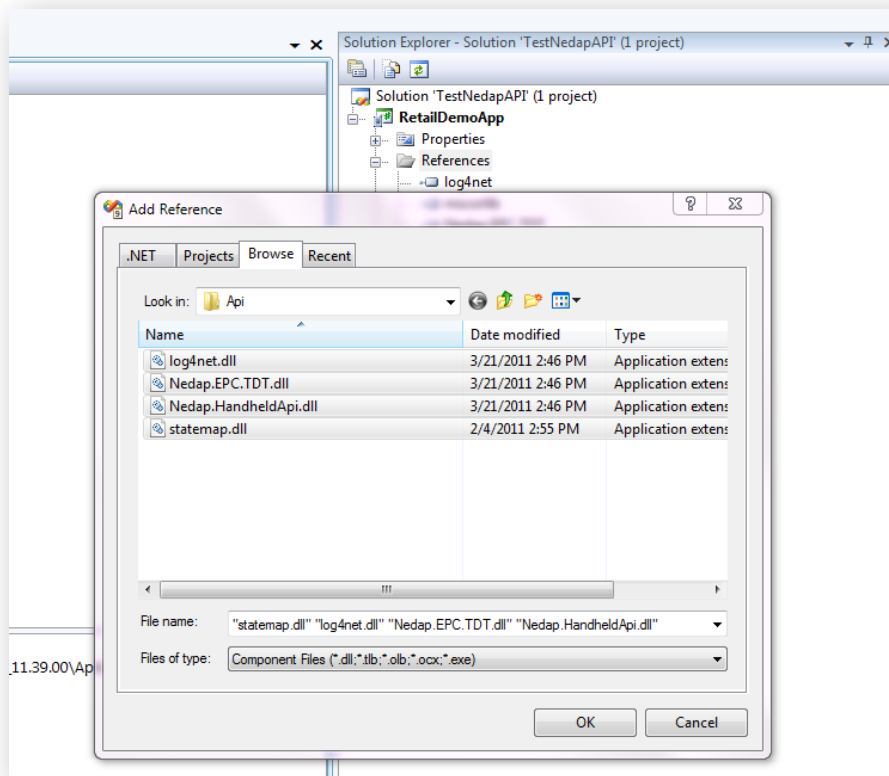
For an example, see the NedapHandheldAPIDemo application.

Windows (in Visual Studio)

When developing application based up on this API there are few things you have to take care of in order to get started. Please note that since there is no standard way in dealing with Bluetooth connecting and pairing under Windows CE or Mobile; this has to be implemented by the software developer. This API assumes a SerialPort that's already paired to the !D Hand.

First you need to add libraries to you project. Libraries that should be added are: InTheHand.Net.Personal.dll, log4net.dll, Nedap.EPC.TDT.dll, Nedap.HandheldApi.dll, statemap.dll. InTheHand.Net.Personal.dll library requires native library 32feetWidcom.dll for Widcomm (Broadcom) Bluetooth stack support, and this library should be added as a file to project and in properties of this file 'Copy to Output Directory' should be set to 'Copy always', and 'Build Action' to 'Content'.





The next step is to create TDTEngine and initialize it. It can be initialized without parameters or with parameters of XMLDefinitions type which is public enumerator in TDTEngine. This list represents translation definitions XML files. When loading all definitions call Initialize without parameters:

```
Nedap.EPC.TDT.TDTEngine engine = new Nedap.EPC.TDT.TDTEngine();
engine.Initialize();
```

When loading only wanted definitions call Initialize with parameters:

```
Nedap.EPC.TDT.TDTEngine engine = new Nedap.EPC.TDT.TDTEngine();
XmlDefinitions[] definitionsList = { XmlDefinitions.SGLN_96,
    XmlDefinitions.SGTIN_96 };
engine.Initialize(definitionsList);
```

Then to use the API, create a SerialCommunicationPort, Session and PresentationFacade:

```
public ICommunicationPort Port { get; private set; }
public Session Session { get; private set; }
public PresentationFacade Facade { get; private set; }
```

To connect with the !D Hand, there are two options: use the 32feet.net library to automatically detect and connect to the !D Hand. If that's not possible, a fallback scenario for using a Bluetooth serial port is also possible. A universal solution would be to try to instantiate BluetoothCommunicationPort, and if platform is not supported an exception would be thrown. This exception can be catch, and fallback to SerialPort class can be done. With this approach more robust solution can be created.

32feet.net library

If the phone uses Microsoft or Widcomm (Broadcom) Bluetooth stack (32feet.net library), device discovery, pairing and connecting (implemented in BluetoothCommunicationPort) can be done without user interaction:

```
Port = new BluetoothCommunicationPort();
Session = new Session(Port, true);
Facade = new PresentationFacade(Session);
```

Serial port fallback

If the phone does not support mentioned Bluetooth stacks, SerialCommunicationPort class can be used. This approach requires the user to use target phone Bluetooth software to manually discover, pair and connect to the !D Hand Bluetooth Serial Port. Port name (portName) is the name of the System.IO.Ports.SerialPort that is used to connect to the handheld reader (e.g. "COM1"). Note that port name can be provided from UI and that user should select COM port used to connect to the !D Hand.

```
Port = new SerialCommunicationPort(portName, 9600, Parity.None, 8, StopBits.One);
Session = new Session(Port, true);
Facade = new PresentationFacade(Session);
```

Port name is name of the System.IO.Ports.SerialPort available on the device (example: "COM1"),

```
public SerialCommunicationPortPort = new SerialCommunicationPort(portName, 9600, Parity.None, 8, StopBits.One);
```

Port is then used to create Session:

```
public Session Session = new Session(Port, true);
```

PresentationFacade is then created with that Session.

```
public PresentationFacade Facade = new PresentationFacade(Session);
```

Communication with device is easily done through PresentationFacade and all device responses and result data is returned through events. Subscribing to event and calling method via presentation:

```
Program.Controller.Facade.OnReadRfidTag += new System.EventHandler<ReadRfidTagEventArgs>(facade_OnReadRfidTag);
Program.Controller.Facade.RfidInventoryStart();
```

When updating GUI on triggered event developer must take care of asynchronous GUI update because of the thread security. Example of solving this problem with BeginInvoke method:

```
private void facade_OnReadRfidTag(object sender, ReadRfidTagEventArgs e)
{
    lbInventoryItemsList.BeginInvoke(new UpdateUIDelegate(FormUpdateMethod),
        e.EpcObservation);
}

private void FormUpdateMethod(EpcObservation epcObservation)
{
    lbInventoryItemsList.Items.Add(epcObservation.EpcCode.PureIdentityURI);
}
```

On application exit, a dispose of classes used for interaction should be done in this order:


```
public void Dispose()
{
    if (Session != null)
    {
        Facade.Dispose();
        Port.Dispose();
        Session.Dispose();
    }
}
```

This sums up basic information and code examples for beginning development with this API. More details can be found in the example application (RetailDemoApp).

A. Frequently Asked Questions

Using Bluetooth and Wi-Fi together used to cause a lot of problems. Isn't that the case as well with the ID Hand, when used together with a mobile device?

This indeed used to be a problem in the past, but it's not anymore. Bluetooth 2.1+EDR introduced some sophisticated algorithms to cope with these issues. Furthermore, most mobile device are able to intelligently use both Bluetooth and Wi-Fi at the same time, by using time multiplexing. In our applications, we didn't see any issues related to this. Consider it a thing of the past.

B. Update firmware

This chapter contains instructions to update the firmware on the !D Hand to the newest version.

What you need:

- !D Hand
- Micro USB cable
- Firmware (!D Hand firmware x.x.x.hex)
- AVRprog.exe (Windows) or avrdude (Mac OS X) - see archive with firmware
- Computer with Windows or Mac OS X (Linux is available upon request)

1. Install the driver for the !D Hand

The driver is available at the following URL:

<http://www.ftdichip.com/Drivers/VCP.htm>

2. Connect the !D Hand to the computer

The right LED lights up orange, to indicate normal charging.



3. Hold down the button for more than 8 seconds

The !D Hand will reboot and go into the bootloader. Both LED's lights up red. The !D Hand will remain 10 seconds in this state.



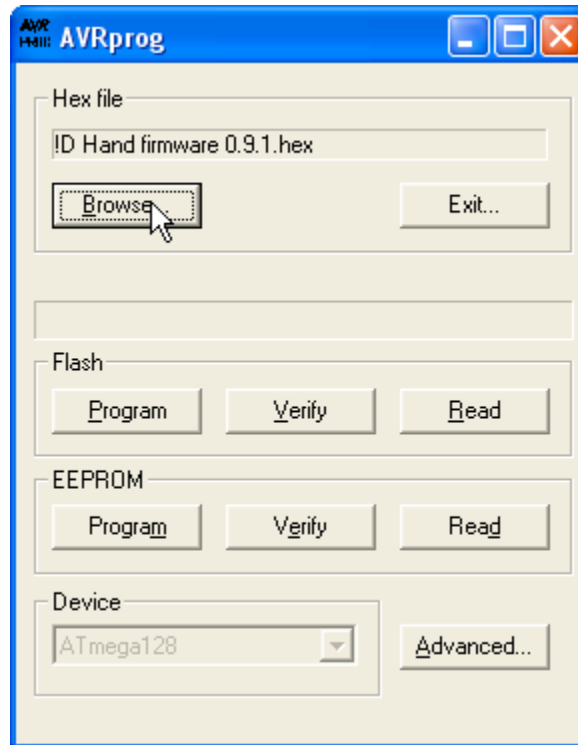
From now on there is a difference in steps between Windows or Mac OS X. The instructions for Windows are first, followed by the instructions for Mac OS X.

Windows 4. Start AVRProg.exe while the !D Hand is in the bootloader (not before, and not after)



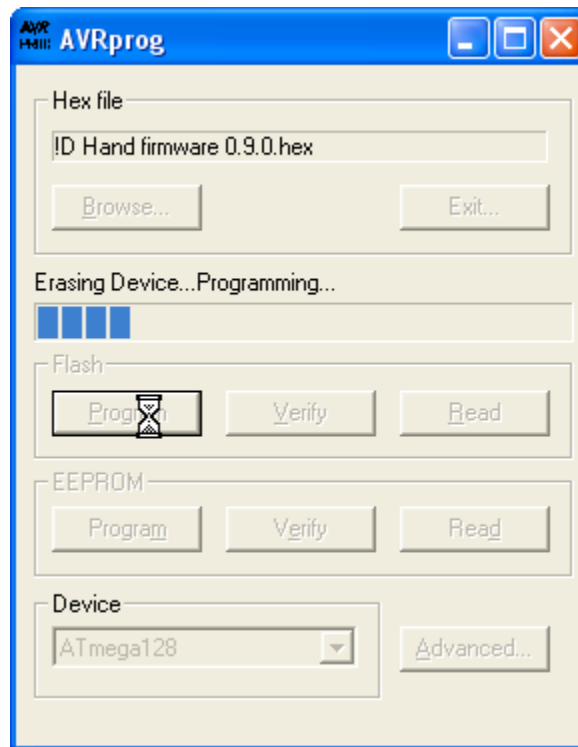
Windows 5. Select the new firmware file

Click on browse and select the firmware file. Click on open.



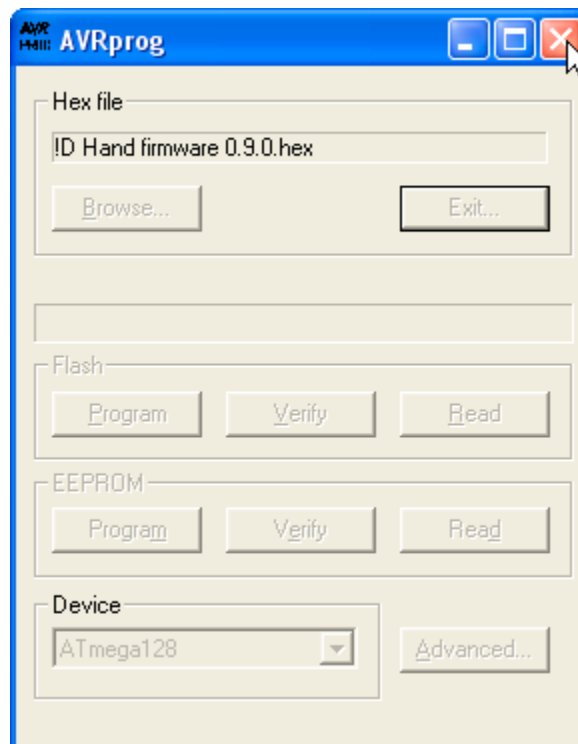
Windows 6. Click on 'Program' in the 'Flash' section

The program will start erasing, upload and verify the new firmware. After the upload is finished the program shows that everything is done OK.



Windows 7. Click on exit to quit the boot loader on the !D Hand

The options get inactive and the !D Hand reboots. After that, it's safe to close the application.



Mac OS X 4. Execute the following command while the !D Hand is in the bootloader (not before, and not after)

```
avrdude -pm1280 -cavr109 -P/dev/tty.usbserial-<serial> -b115200 -Uflash:w:\!D\ Hand\ firmware\ <version>.hex:a
```

Replace <serial> and <version> with the appropriate variables. The version depends on the file in the firmware archive. The serial number can be found by executing the following command:

```
ls /dev/tty.usbserial*
```

The result is then:

```
/dev/tty.usbserial-1012
```

For example for serial 1012 and version 0.9.1:

```
avrdude -pm1280 -cavr109 -P/dev/tty.usbserial-1012 -b115200 -Uflash:w:\!D\ Hand\ firmware \ 0.9.1.hex:a
```

The result should be:

```
Connecting to programmer: .
Found programmer: Id = "AVRBOOT"; type = S
Software Version = 0.8; No Hardware Version given.
Programmer supports auto addr increment.
Programmer supports buffered memory access with buffersize=256 bytes.

Programmer supports the following devices:
    Device code: 0x43

avrdude: AVR device initialized and ready to accept instructions

Reading | ##### | 100% 0.00s

avrdude: Device signature = 0x1e9703
avrdude: NOTE: FLASH memory has been specified, an erase cycle will be performed

    To disable this feature, specify the -D option.

avrdude: erasing chip
avrdude: reading input file "!D Hand firmware 0.9.1.hex"
avrdude: input file !D Hand firmware 0.9.1.hex auto detected as Intel Hex
avrdude: writing flash (49568 bytes):

Writing | ##### | 100% 5.81s

avrdude: 49568 bytes of flash written
avrdude: verifying flash memory against !D Hand firmware 0.9.1.hex:
avrdude: load data flash data from input file !D Hand firmware 0.9.1.hex:
```

```
avrdude: input file !D Hand firmware 0.9.1.hex auto detected as Intel Hex
avrdude: input file !D Hand firmware 0.9.1.hex contains 49568 bytes
avrdude: reading on-chip flash data:
```

```
Reading | ##### | 100% 4.66s
```

```
avrdude: verifying ...
avrdude: 49568 bytes of flash verified
```

```
avrdude done. Thank you.
```

If the !D Hand is not in bootloader mode (both red LEDs are not on), the error will be:

```
Connecting to programmer: .avrdude: butterfly_recv(): programmer is not responding
```

Hold the button for more than ten seconds and try again.

C. Licensing

The embedded software in the !D Hand uses code from the Arduino project - LGPL licensed code. To comply with the license agreement, we provide you with the following:

- Release changes to the libraries covered by the LGPL. This code is available at <https://github.com/nedap/arduino>. Upon request, we can send this code on a physical device. Shipping and handling cost may apply.
- On request, we provide binary object files that allow for the relinking of the firmware against updated versions of the LGPL code. Please send this request to rfidretail@nedap.com.

GNU LESSER GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. <<http://fsf.org/>>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

0. Additional Definitions.

As used herein, “this License” refers to version 3 of the GNU Lesser General Public License, and the “GNU GPL” refers to version 3 of the GNU General Public License.

“The Library” refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An “Application” is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A “Combined Work” is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the “Linked Version”.

The “Minimal Corresponding Source” for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The “Corresponding Application Code” for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

- a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or
- b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

- a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.
- b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

- a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.
- b) Accompany the Combined Work with a copy of the GNU GPL and this license document.
- c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.
- d) Do one of the following:

0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.

1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.

e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

- a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.
- b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

D. Change notes

version	date	changes
0.9	january 10, 2011	[DH] First public release.
0.9.1	february 13, 2011	[DH] Added Windows function calls and description [DH] Added information on how to get started with Windows development [DH] Added function to retrieve device information [DH] Updated device information command in transport layer to add model number (SKU) [DH] Fixed command 'SetEPCParameters' in transport layer; values for settings were wrong [DH] Removed all references to set the maximum EPC length; now defaults to read maximum lengths of 496 bits [DH] Added functionality to do high-volume inventory (using Session 2 and Session 3) [DH] Added instructions on how to update the firmware of the !D Hand [DH] Added license information on the software used on the !D Hand [DH] Fixed length error in the transport layer 'Vibration' command example [DH] Fixed error in the transport layer 'SetEPCParameters' command: the order in the set Q value message was not right
0.9.2	february 25, 2011	[DH] Added information on how to update the firmware. [DH] Added vibration tune command in all layers. [DH] Added new write EPC command in the transport layer that checks if just one tag is in the field, writes the new EPC to a tag (automatically adjusting PC words) and verifies what's written [DH] Added information on how to update the firmware on Mac OS X
1.0	march 22, 2011	[DH] Changed documentation for Xcode 4 [DH] Fixed small errors
1.1	june 17, 2011	[DH] Add 'large packet' with type 0x07 [DH] Change naming of Windows events OnDidDisconnect and OnDidConnect to new names OnDisconnect and OnConnect [DH] Changed naming of some Windows methods to be more consistent with what is 'normal' [DH] Added documentation for using the 32feet.net Bluetooth library