Identive Infrastructure

NFC Wrapper - Reference manual

Version 1.2
2012-02-02

Keywords

NFC, NDEF, SCM_NFC.dll, P2P

Abstract

This document describes how to use the NFC Wrapper.

Audience

This document is for the technical staff of Identive and for developers who want to use the NFC Wrapper in their own projects.

Revision history

Rev	Date	Author	Description
1.0	2009-03-11	MDR	Initial version
1.1	2011-08-26	MDR	Description of the new features of the NFC Wrapper V1.1 added
1.2	2012-02-02	MDR	Document rebranded

Contact information

For additional information, please visit http://www.identive-infrastructure.com/

Content

1 Legal information

1.1 Disclaimer

The content published in this document is believed to be accurate. Identive does not, however, provide any representation or warranty regarding the accuracy or completeness of its content and regarding the consequences of the use of information contained herein. If this document has the status “Draft”, its content is still under internal review and yet to be formally validated.

Identive reserves the right to change the content of this document without prior notice. The content of this document supersedes the content of previous versions of the same document. The document may contain application descriptions and/or source code examples, which are for illustrative purposes only. Identive gives no representation or warranty that such descriptions or examples are suitable for the application that the reader may want to use them for.

Should you notice problems with the provided documentation, please provide your feedback to support@identive-infrastructure.com.

1.2 Licenses

The source code examples contained in this document are provided for illustrative purposes only and subject to the following restrictions:

- You may at your own risk use or modify the source code provided in the document in applications you may develop.
- You must not copy or distribute parts of or the entire source code without prior written consent from Identive.
- You must not combine or distribute the source code provided with Open Source Software or with software developed using Open Source Software in a manner that subjects the source code or any portion thereof to any license obligations of such Open Source Software.

If the document contains technical drawings related to Identive products, they are provided for documentation purposes only. Identive does not grant you any license to its designs.

1.3 Trademarks

MIFARE is a registered trademark of NXP Semiconductors BV.
FeliCa is a registered trademark of Sony Corporation.
Windows is a registered trademark of Microsoft Corporation.

2 Introduction

The NFC Wrapper is implemented as a 32 bit native Windows library named SCM_NFC.dll.

The NFC Wrapper hides the hardware specific differences between all kinds of NFC devices, NFC tags and NFC Forum enabled tags. It provides a unique API to the application developer, which enables the developer to read and modify NDEF records without further knowledge of the underlying hardware.

The API provided by the NFC Wrapper is described in this manual.

2.1 Functional principle

When activated, the NFC Wrapper continuously monitors all locally connected NDEF compatible devices (e.g. SDI010, SCL010, SCL3711 etc.) for the presence of NFC devices, NFC tags and NFC Forum enabled tags.

If an NFC tag is detected, the library automatically establishes a communication channel, reads the NDEF data from the tag, closes the connection, adds the NDEF data together with the sender address to an internal message queue, and notifies the application.

The application can read the NDEF messages in the message queue at any time. A message that has been read is automatically removed from the message queue by the library.

When modifying the NDEF data on a tag or NFC device, the application just sends back the new NDEF record to the sender address. The NFC Wrapper will wait for the tag for the specified timeout interval and will then modify the data on the tag. If the application specifies a broadcast address, the library will modify the first tag that will be detected within the timeout interval. While waiting for a tag (or a timeout), the application is blocked.

2.2 Supported operating systems

32 bit versions of Windows® 2000, XP, Vista, or 7

2.3 Supported devices

The NFC Wrapper only supports devices made by Identive.
The current version of the NFC Wrapper actually supports the devices SCL3711, SCL010, SDI010, SCL011 and SDI011 (default devices).

Support for future devices can be added by inserting the product IDs of these devices into the NFC Wrapper's Ini file (SCM_NFC.ini). Add the following two lines to the Ini file.

[DEVICES]
Supported = <PID1>, <PID2>, … , <PIDn>

The variables <PID?> have to be replaced with the product IDs of the devices to be supported.
The product IDs have to be specified in decimal format (e.g. 21137 for SCL010).

The Ini file is located in the same folder as the NFC Wrapper (<WindowsDir>\system32 by default).
The Ini file is not required if the NFC Wrapper should support the default devices only.

2.4 Supported smart cards

NFC Type 1 Tags
Topaz (Innovision)

NFC Type 2 Tags
MIFARE Ultralight (NXP), MIFARE Ultralight C (NXP)

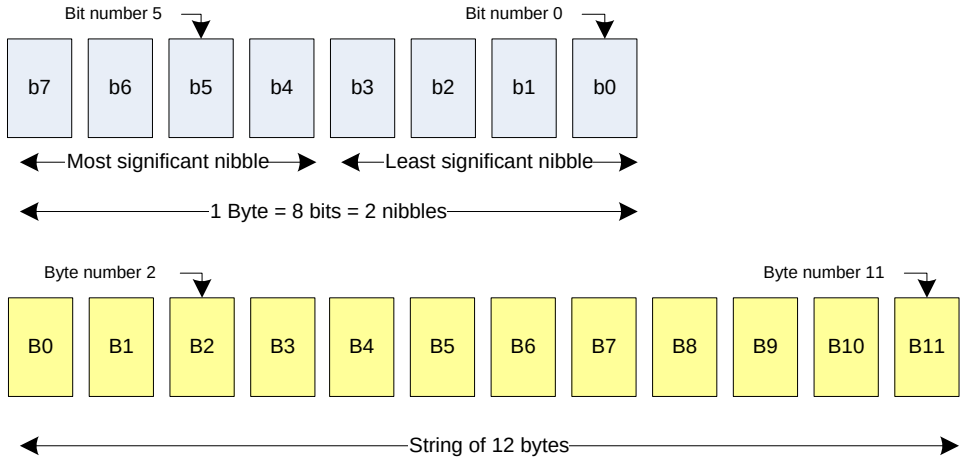
NFC Type 3 Tags
FeliCa (Sony)

NFC Type 4 Tags
MIFARE DESFire (NXP), MIFARE DESFire-EV1 (NXP)

NFC Forum Formatted Tags
MIFARE Classic 1K (NXP), MIFARE Classic 4K (NXP), MIFARE Plus in SL1 (NXP)

2.5 Conventions

Bits are represented by lower case ‘b’ where followed by a numbering digit.
Bytes are represented by upper case ‘B’ where followed by a numbering digit.



Examples
The decimal number 163 is represented

- in hexadecimal as 0xA3
- in binary as (10100011)b

The least significant nibble of 0xA3 is

- 0x3 in hexadecimal
- (0011)b in binary

The most significant nibble of =0xA3 is

- 0xA in hexadecimal
- (1010)b in binary

3 NFC Wrapper structures

The exported functions of the SCM_NFC.DLL use the following structures.

3.1 NDEFAddress

This structure contains the unique address of a NFC device, NFC tag or NFC Forum enabled tag.

```
typedef struct _NDEFAddress {
  byte[12] rgbyAddress;
} NDEFAddress,
 *LPNDEFAddress;
```

Members

`rgbyAddress`

A 12 byte identifier that identifies a sender or a recipient of an NDEF message.
The address is the concatenation of a two byte card reader identifier (first two bytes) and a ten byte card or NFC device identifier. The card ID is the UID of the card filled with preceding zeros if necessary.
When writing NDEF data to a tag or NFC device, the address is used by the NFC Wrapper to redirect the NDEF message to the correct target.
If the address consists of 12 zeros, then it is a broadcast address.

3.2 MESSAGEDETAILS

This structure contains the details of a received NDEF message.

```
typedef struct _MessageDetails {
  char[64]   szDeviceName,
  DWORD      dwSenderType,
  SYSTEMTIME Timestamp;
} MessageDetails,
 *LPMessageDetails;
```

Members

`szDeviceName`

The name of the device that has received the NDEF message. This may for instance be the PC/SC name of the local card reader.

`dwSenderType`

The type of the NFC device, NFC tag or NFC Forum enabled tag which has sent the NDEF message.
If the top most bit is set, then the sender is a NFC device, otherwise the sender is a NFC tag or NFC Forum enabled tag. The other 31 bits are used to encode the actual type of the NFC device or tag (see 6.3 for the list of defined types).

`TimeStamp`

The system time at the moment when the message was received.

4 NFC Wrapper notification messages

The following messages may be sent by SCM_NFC.DLL to the calling application.

4.1 WM_NFC_NOTIFY

This message is used to notify the application that an internal state of the NFC Wrapper library has been changed. The parameter wParam contains the ID of the event that happened.

Message-ID

WM_USER + 1

wParam

`NFC_NDEF_FOUND`

Sent after a new NDEF messages has been added to the message queue.
lParam contains the size of the new message. The message size might be null, if the NDEF message is empty. Even empty NDEF messages will be added to the message queue.

`NFC_DEVICE_CHANGED`

Sent after the number of NDEF compliant devices has been changed (e.g. after a card reader has been connected to or removed from the system).
lParam contains the new number of present NDEF compatible devices.

`NFC_CONNECTED`

Sent after a possible NFC device or NFC tag or NFC Forum enabled tag has been detected but before it is accessed. This message may be used by the application to display a “reading in progress” message.

`NFC_DISCONNECTED`

Sent if an NFC device or NFC tag or NFC Forum enabled tag that was present can no longer be detected (e.g if a tag was removed from the reader).

`NFC_UNKNOWN_SERVICE`

Sent if the detected device or tag is not a NFC device or NFC tag or NFC Forum enabled tag (e.g. if a Mifare card does is not formatted to hold NDEF data). This message may be used by the application to display an “invalid tag” message.

`NFC_IDLE`

Similar to `NFC_DISCONNECTED`. This message is sent, when an NFC device or tag is removed from a reader and if there is no other NFC device or tag present on any other connected reader. In case that only one reader is connected, `NFC_IDLE` is the same as `NFC_DISCONNECTED`.

5 NFC Wrapper functions

The following functions are exported by the SCM_NFC.DLL.

5.1 Initialize

This function initializes the NFC Wrapper library. This is always the first function to call. The library will not interact with any NFC devices, NFC tags or NFC forum enabled tags as long as it is has not been initialized and all attempts to call any of the other functions will result in an ERR_NOT_INITIALIZED error.

```
DWORD WINAPI Initialize (
  __in_opt  HANDLE hWnd
);
```

Parameters

`hWnd`

The handle of the application's main window.
The DLL uses this handle to send notification messages to the calling application.
This parameter may be null.

Return Codes

This function always returns `ERR_SUCCESS`.

5.2 StartListening

After calling this function, the library starts to scan for the specified NFC devices or tags.

```
DWORD WINAPI StartListening ( void );
```

Return Codes

`ERR_SUCCESS`	On successful execution
`ERR_NOT_INITIALIZED`	If the library has not been initialized.

5.3 StopListening

This function stops the NFC Wrapper library from scanning for NFC devices or tags. Calling this function will not change the state of the message queue. It will especially not remove any messages from the queue.

```
DWORD WINAPI StopListening ( void );
```

Return Codes

`ERR_SUCCESS`	On successful execution
`ERR_NOT_INITIALIZED`	If the library has not been initialized.

5.4 Rescan

This function scans for tags on all connected card readers and reads the tags once again. The read data (if any) will be added to the messages queue.

```
DWORD WINAPI Rescan ( void );
```

Return Codes

ERR_SUCCESS	On successful execution
ERR_NOT_INITIALIZED	If the library has not been initialized.
ERR_NOT_LISTENING	If listening is not active (StartListening was not yet called, or StopListening was called afterwards).

5.5 SetTarget

This function can be used to change the NFC tags and devices supported by the Wrapper. If, for instance, the support of NFC devices is disabled, the Wrapper will not scan for NFC devices anymore. Disabling unsupported targets can increase the performance of the system.
By default, the NFC Wrapper scans for all supported NFC tags, but not for NFC devices.

```
DWORD WINAPI SetTarget (
    __in      DWORD dwTarget,
    __in_opt  LPDWORD lpOldTarget
);
```

Parameters

dwTarget

The NFC tags and devices to be supported
This parameter is a disjunction of the constants defined in chapter 6.3.
This parameter is mandatory.

lpOldTarget

The previous value of the NFC tags and devices to be supported
This parameter is a disjunction of the constants defined in chapter 6.3.
This parameter may be null.

Return Codes

If the library is not initialized, the function returns ERR_NOT_INITIALIZED, otherwise it returns ERR_SUCCESS.

Remark

To enable peer to peer communication with NFC devices, call SetTarget with the parameter NFC_TAGS_AND_DEVICES.

5.6 GetNDEFQueueInfo

This function returns some status information about the NDEF message queue.

```
DWORD WINAPI GetNDEFQueueInfo (
  __out_opt LPDWORD lpDeviceCount,
  __out_opt LPDWORD lpMessageCount,
  __out_opt LPDWORD lpNextMessageSize,
);
```

Parameters

`lpDeviceCount`

The number of present devices which are able to receive NDEF messages.
This parameter may be null.

`lpMessageCount`

The number of NDEF messages waiting in the message queue.
This parameter may be null.

`lpNextMessageSize`

The size in bytes of the next NDEF message in the message queue.
This parameter may be null.

Return Codes

If the library is not initialized, the function returns ERR_NOT_INITIALIZED, otherwise it returns ERR_SUCCESS.

5.7 ClearNDEFQueue

This function either removes all messages or just the messages of a certain sender from the NDEF message queue.

```
DWORD WINAPI ClearNDEFQueue (
  __in_opt LPNDEFAddress lpNDEFAddress
);
```

Parameters

`lpNDEFAddress`

If this parameter is not null, only the messages from this sender will be removed from the queue, otherwise all messages will be removed. A broadcast address will also remove all messages
This parameter may be null.

Return Codes

If the library is not initialized, the function returns ERR_NOT_INITIALIZED, otherwise it returns ERR_SUCCESS.

5.8 ReadNDEF

This function returns the oldest NDEF message from the message queue and removes it from the queue.

```
DWORD WINAPI ReadNDEF(
  __out     LPNDEFADDRESS lpNDEFAddress,
  __out_opt LPMESSAGEDETAILS lpMessageDetails,
  __out     PByte lprgbyNDEF,
  __in_out  LPDWORD lpNDEFSize
);
```

Parameters

`lpNDEFAddress`

The address of the NFC device or tag that has sent the NDEF data (see 3.1).
The application is responsible to allocate and free the memory.
This parameter is mandatory.

`lpMessageDetails`

The message details (see 3.2).
This parameter may be null.
The application is responsible to allocate and free the memory.

`lprgbyNDEF`

A pointer to a memory area in which the NDEF data will be stored.
The application is responsible to allocate and free the memory.
Call GetNDEFQueueInfo to get the size of the next NDEF message in the queue.
This parameter is mandatory.

`lpNDEFSize`

A pointer to a variable that contains the amount of memory allocated for `lprgbyNDEF`. When the function `ReadNDEF` returns, this variable contains the size of the returned NDEF message.
This parameter is mandatory.

Return Codes

`ERR_SUCCESS`	On successful execution
`ERR_NOT_INITIALIZED`	If the library has not been initialized.
`ERR_NO_MESSAGE`	If the message queue is empty and the library is initialized.
`ERR_INVALID_PARAMETER`	If one of the mandatory parameters is null.
`ERR_BUFFER_TOO_SMALL`	If *`lpNDEFSize` is less than the size of the next message in the queue. The required memory to store the NDEF data is returned in *`lpNDEFSize`.

5.9 WriteNDEF

This function sends an NDEF message to a NFC device, NFC tag or NFC Forum enabled tag.

```
DWORD WINAPI WriteNDEF(
  __in_opt LPNDEFADDRESS lpNDEFAddress,
  __out_opt LPMESSAGEDETAILS lpMessageDetails,
  __in_opt PByte lprgbyNDEF,
  __in_out LPDWORD lpNDEFSize,
  __in     BOOL bDontOverwrite,
  __in     BOOL bFormatTag,
  __in     DWORD dwTimeOut
);
```

Parameters

`lpNDEFAddress`

The address of the NFC device or tag that should receive the NDEF data (see 3.1).
The application is responsible to allocate and free the memory.
This parameter may be null.
If this parameter is either null or a broadcast address, then the NDEF data will be written to the first NFC device or tag that will be detected. Otherwise only the NFC device or tag with the specified address will be modified and all other tags will be ignored.

`lpMessageDetails`

The message details (see 3.2). This structure will be filled by the library after successfully writing the NDEF data to a NFC device or NFC tag or NFC Forum enabled tag.
This parameter may be null.
The application is responsible to allocate and free the memory.

`lprgbyNDEF`

A pointer to a memory area in which the NDEF data is stored.
The application is responsible to allocate and free the memory.
This parameter is optional.
If this parameter is null, then an empty NDEF record will be written to the NFC device or tag (if the flags `bDontOverwrite` and `bFormatTag` are set accordingly).

`lpNDEFSize`

A pointer to a variable that contains the size of the NDEF data.
This parameter is optional.
If this parameter is `NULL` or zero, then an empty NDEF record will be written to the NFC device or tag regardless of the content of `lprgbyNDEF` (but only, if the flags `bDontOverwrite` and `bFormatTag` are set accordingly).
If `lpNDEFSize` is not `NULL` and if `*lpNDEFSize` is larger than the maximum storage capacity of the NFC device or tag, then the maximum allowed NDEF size is returned in `*lpNDEFSize`. The returned value may be different for different values of the flag `bFormat`.

`bDontOverwrite`

Set this parameter to TRUE, if existing NDEF records must not be modified. In this case, the NDEF record will only be written, if the tag contains either an empty NDEF record or if it has not yet been formatted as an NDEF tag or NDEF Forum enabled tag.

`bFormatTag`

Set this parameter to TRUE, if the NFC Wrapper should try to format an unformatted tag as NDEF tag or NDEF Forum enabled tag. In case of a Mifare Standard card, this flag must be set to TRUE if the NFC Wrapper should try to allocate unused card sectors of a multi application card, if necessary (if the NDEF will not fit into the currently used sectors).

`dwTimeOut`

The time in seconds that the library should wait for the tag with the specified address.
If no tag is detected within this interval, the function returns ERR_TIMEOUT and the NDEF data will not be written.
The calling thread is blocked until the function returns.

Return Codes

`ERR_SUCCESS`	On successful execution
`ERR_NOT_INITIALIZED`	If the library has not been initialized.
`ERR_INVALID_PARAMETER`	If one of the mandatory parameters is null.
`ERR_TIMEOUT`	If the specified tag was not found within `dwTimeOut`.
`ERR_WRITE_ERROR`	If a communication error occurs during the write process.
`ERR_NDEF_TOO_LARGE`	If the NDEF data is larger than the space on the tag.
`ERR_TAG_NOT_SUPPORTED`	If the presented tag is not supported by the library.
`ERR_TAG_NOT_FORMATTED`	If `bFormatCard` is not specified and the tag is not formatted.
`ERR_TAG_NOT_EMPTY`	If `bDontOverwrite` is specified and the tag cannot be formatted.
`ERR_TAG_READONLY`	If the tag is write protected.
`ERR_CANCELLED`	If waiting for a tag is cancelled before the timeout interval elapses.

Note

If a communication error occurs while writing the NDEF data to the tag or NFC device, then the library will automatically try to resend the NDEF data as long the timeout interval has not elapsed. If it finally fails, it returns `ERR_WRITE_ERROR`.

5.10 NDEF2XML

This function converts an NDEF message into XML.

```
DWORD WINAPI NDEF2XML(
    __in     PByte lprgbyNDEF,
  __in     LPDWORD lpNDEFSize,
  __in     PChar lprgchXML,
  __in_out LPDWORD lpXMLSize
);
```

Parameters

`lprgbyNDEF`

A pointer to a memory area in which the NDEF data is stored.
The application is responsible to allocate and free the memory.
This parameter is mandatory.

`lpNDEFSize`

A pointer to a variable that contains the size of the NDEF data.
This parameter is mandatory.

`lprgchXML`

A pointer to a memory area in which the XML data will be stored. The data is not zero terminated. The actual size of the data is returned in `lpXMLSize`.
The application is responsible to allocate and free the memory.
This parameter is optional. If this parameter is NULL, only the size of the XML data is returned in `lpXMLSize`.

`lpXMLSize`

A pointer to a variable that contains the size of the XML data. If the function succeeds, this parameter contains the size of the returned XML data. If the function returns `ERR_BUFFER_TOO_SMALL`, this parameter contains the required minimum size of `lprgchXML`.
This parameter is mandatory.

Return Codes

`ERR_SUCCESS`	On successful execution
`ERR_NOT_INITIALIZED`	If the library has not been initialized.
`ERR_INVALID_PARAMETER`	If one of the mandatory parameters is null.
`ERR_BUFFER_TOO_SMALL`	If `lpXMLSize` is less than the size of the resulting XML data.
`ERR_INVALID_NDEF`	If the NDEF data is invalid and cannot be converted to XML

Note

The payload of NDEF records is returned either as XML or as plain text or as base64 encoded text.
Only NFC well-knows NDEF records are returned as XML. Payloads of records of a MIME type that starts with “text” or “application/text” are returned as plain text. All other payloads are returned as base64 encoded text.

5.11 CreateNDEFText

This function creates an NDEF message with a single Smart Poster NDEF record.

```
DWORD WINAPI CreateNDEFSp(
  __in     PChar szText,
  __in_opt PChar szLanguage,
  __out    PByte lprgbyNDEF,
  __in_out LPDWORD lpNDEFSize,
);
```

Parameters

`szText`

A pointer to a zero terminated string that contains the text.
This parameter must only contain ASCII characters. Non-ASCII characters have to be UTF-8 encoded by the application. This function will automatically remove unnecessary white spaces from the text as defined in the NDEF record specification.
This parameter is mandatory and it must not be an empty string.

`szLanguage`

A pointer to a zero terminated string that contains the IANA language code of the text. See RFC3066 for further information.
This parameter is optional. If `szLanguage` is not specified, the default language code "en-US" is inserted in the resulting NDEF text record.

`lprgbyNDEF`

A pointer to a memory area in which the resulting NDEF message will be stored.
The application is responsible to allocate and free the memory.
This parameter is mandatory.

`lpNDEFSize`

A pointer to a variable that contains the amount of memory allocated for `lprgbyNDEF`. When the function returns, this variable contains the size of the created NDEF message. If the function returns `ERR_BUFFER_TOO_SMALL`, this parameter contains the required minimum size of `lprgbyNDEF`.
This parameter is mandatory.

5.12 CreateNDEFURI

This function creates an NDEF message with a single Smart Poster NDEF record.

```
DWORD WINAPI CreateNDEFSp(
   __in     PChar szURI,
  __out     PByte lprgbyNDEF,
  __in_out LPDWORD lpNDEFSize,
);
```

Parameters

`szURI`

A pointer to a zero terminated string that contains the URI.
This parameter must only contain ASCII characters. Non-ASCII characters have to be UTF-8 encoded by the application.
This parameter is mandatory and it must not be an empty string.

`lprgbyNDEF`

A pointer to a memory area in which the resulting NDEF message will be stored.
The application is responsible to allocate and free the memory.

This parameter is mandatory.

`lpNDEFSize`

A pointer to a variable that contains the amount of memory allocated for `lprgbyNDEF`. When the function returns, this variable contains the size of the created NDEF message. If the function returns `ERR_BUFFER_TOO_SMALL`, this parameter contains the required minimum size of `lprgbyNDEF`.
This parameter is mandatory.

5.13 CreateNDEFSp

This function creates an NDEF message with a single Smart Poster NDEF record.

```
DWORD WINAPI CreateNDEFSp(
  __in     PChar szURI,
  __in_opt PChar szComment,
  __in_opt PChar szLanguage,
  __in_opt PByte lpbyAction,
  __in opt PDWORD lpSize,
  __in_opt PChar szTargetType,
  __out    PByte lprgbyNDEF,
  __in_out LPDWORD lpNDEFSize,
);
```

Parameters

`szURI`

A pointer to a zero terminated string that contains the URI.
This parameter must only contain ASCII characters. Non-ASCII characters have to be UTF-8 encoded by the application.
This parameter is mandatory and it must not be an empty string.

`szComment`

A pointer to a zero terminated string that contains a description of the URI.
This parameter must only contain ASCII characters. Non-ASCII characters have to be UTF-8 encoded by the application.
This parameter is optional.

`szLanguage`

A pointer to a zero terminated string that contains the IANA language code of the parameter `szComment`. See RFC3066 for further information.
This parameter is optional. If `szLanguage` is not specified, but `szComment` is specified, the default language code “en-US” is inserted in the resulting Smart Poster record.

`lpbyAction`

A pointer to a byte that contains the default action of the Smart Poster record.
This parameter is optional.

`lpSize`

A pointer to a DWORD that contains the size of the object referred by the URI.
This parameter is optional.

`szTargetType`

A pointer to a zero terminated string that contains the MIME type of the object referred by the URI.
This parameter is optional.

`lprgbyNDEF`

A pointer to a memory area in which the resulting NDEF message will be stored.
The application is responsible to allocate and free the memory.
This parameter is mandatory.

`lpNDEFSize`

A pointer to a variable that contains the amount of memory allocated for `lprgbyNDEF`. When the function returns, this variable contains the size of the created NDEF message. If the function returns `ERR_BUFFER_TOO_SMALL`, this parameter contains the required minimum size of `lprgbyNDEF`.
This parameter is mandatory.

Return Codes

`ERR_SUCCESS`	On successful execution
`ERR_NOT_INITIALIZED`	If the library has not been initialized.
`ERR_INVALID_PARAMETER`	If one of the mandatory parameters is null.
`ERR_BUFFER_TOO_SMALL`	If `lpNDEFSize` is less than the size of the resulting NDEF message.

5.14 CreateNDEFvCard

This function creates an NDEF message with a single NDEF record containing a vCard.

```
DWORD WINAPI CreateNDEFvCard(
  __in     PChar szVCard,
  __out    PByte lprgbyNDEF,
  __in_out LPDWORD lpNDEFSize,
);
```

Parameters

`szVCard`

A pointer to a zero terminated string that contains the vCard data. The vCard data is not verified by the function to be valid. The application is responsible to ensure that only valid vCard data is passed to this function.
This parameter is mandatory and it must not be an empty string.
See RFC2426 for more information about the vCard data format.

`lprgbyNDEF`

A pointer to a memory area in which the resulting NDEF message will be stored.

The application is responsible to allocate and free the memory.
This parameter is mandatory.

`lpNDEFSize`

A pointer to a variable that contains the amount of memory allocated for `lprgbyNDEF`. When the function returns, this variable contains the size of the created NDEF message. If the function returns `ERR_BUFFER_TOO_SMALL`, this parameter contains the required minimum size of `lprgbyNDEF`.
This parameter is mandatory.

Return Codes

`ERR_SUCCESS`	On successful execution
`ERR_NOT_INITIALIZED`	If the library has not been initialized.
`ERR_INVALID_PARAMETER`	If one of the mandatory parameters is null.
`ERR_BUFFER_TOO_SMALL`	If `lpNDEFSize` is less than the size of the resulting NDEF message.

6 Constants

6.1 Error Codes

This is the list of the currently defined error codes which might be returned by the functions of the NFC Wrapper.

```
ERR_SUCCESS              0x00
ERR_NOT_INITIALIZED      0x01
ERR_INVALID_PARAMETER    0x02
ERR_BUFFER_TOO_SMALL     0x03
ERR_NO_MESSAGE           0x04
ERR_TIMEOUT              0x05
ERR_WRITE_ERROR          0x06
ERR_NDEF_TOO_LARGE       0x07
ERR_TAG_NOT_SUPPORTED    0x08
ERR_TAG_NOT_FORMATTED    0x09
ERR_TAG_NOT_EMPTY        0x0A
ERR_TAG_READONLY         0x0B
ERR_CANCELLED            0x0C
ERR_INVALID_NDEF         0x0D
ERR_NOT_LISTENING        0x0E
```

6.2 Message Codes

The following message codes are used as `wParam` of the `WM_NFC_NOTIFY` message (see also 4.1).

```
NFC_NDEF_FOUND           0x01
NFC_DEVICE_CHANGED       0x02
NFC_UNKNOWN_SERVICE      0x03
NFC_CONNECTED            0x04
NFC_DISCONNECTED         0x05
NFC_IDLE                 0x06
```

6.3 Supported NFC devices and tags

The following values are already defined. Other values may be added in future as needed.

```
TAG_UNKNOWN              0x00000000;
TAG_MIFARE_STD_1K        0x00000001;
TAG_MIFARE_STD_4K        0x00000002;
TAG_MIFARE_UL            0x00000004;
TAG_MIFARE_DESFIRE       0x00000008;
TAG_FELICA               0x00000010;
TAG_TOPAZ                0x00000020;
TAG_SMARTMX              0x00000040;
TAG_MIFARE_DESFIRE_EV1   0x00000080;
TAG_MIFARE_PLUS          0x00000100;
TAG_MIFARE_UL_C          0x00000200;

DEVICE_GENERAL           0x80000000;

NFC_TAG_TYPE1            TAG_TOPAZ;
NFC_TAG_TYPE2            TAG_MIFARE_UL | TAG_MIFARE_UL_C;
```

```
NFC_TAG_TYPE3           TAG_FELICA;
NFC_TAG_TYPE4           TAG_MIFARE_DESFIRE | TAG_MIFARE_DESFIRE_EV1 |
                        TAG_SMARTMX;
NFC_FORMATTED_TAG       TAG_MIFARE_STD_1K | TAG_MIFARE_STD_4K |
                        TAG_MIFARE_PLUS;
NFC_DEVICE              DEVICE_GENERAL;

NFC_TAGS                NFC_TAG_TYPE1 | NFC_TAG_TYPE2 | NFC_TAG_TYPE3 |
                        NFC_TAG_TYPE4 | NFC_FORMATTED_TAG;

NFC_TAGS_AND_DEVICES    NFC_TAGS | NFC_DEVICE;
```

7 XML Document Type Definition

The XML data returned by the function NDEF2XML is formatted according to the following document type definition.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
  <!DOCTYPE NDEF_message [
    <!ELEMENT NDEF_message           (NDEF_record)+>
    <!ATTLIST NDEF_message
      generator                     CDATA #IMPLIED
    >
    <!ELEMENT NDEF_record            (Type,ID,Payload)>
    <!ELEMENT Type                   (#PCDATA)>
    <!ELEMENT ID                     (#PCDATA)>
    <!ELEMENT Payload
(#PCDATA|NDEF_URI|NDEF_Text|NDEF_SmartPoster|NDEF_GenericControl|NDEF_HandoverRe
quest|NDEF_HandoverSelect|NDEF_HandoverCarrier|SmartPoster_action|SmartPoster_si
ze|SmartPoster_type|GenericControl_target|GenericControl_action|GenericControl_d
ata|AlternativeCarrier)*>
    <!ELEMENT NDEF_URI               (URI)>
    <!ELEMENT NDEF_Text              (Langcode,Text)>
    <!ELEMENT NDEF_SmartPoster       (NDEF_record)>
    <!ELEMENT NDEF_GenericControl    (SequenceControl,NDEF_record)>
    <!ELEMENT NDEF_HandoverRequest   (Version,NDEF_record)>
    <!ELEMENT NDEF_HandoverSelect    (Version,NDEF_record)>
    <!ELEMENT NDEF_HandoverCarrier
(CarrierTypeFormat,CarrierType,CarrierData,NDEF_record)>
    <!ELEMENT GenericControl_target  (NDEF_Text)>
    <!ELEMENT GenericControl_action  (ActionFlag,(ActionID|NDEF_record))>
    <!ELEMENT GenericControl_data    (ControlData)>
    <!ELEMENT SmartPoster_action     (Action)>
    <!ELEMENT SmartPoster_size       (Size)>
    <!ELEMENT SmartPoster_type       (Type)>
    <!ELEMENT AlternativeCarrier     (CarrierPowerState,CarrierDataReference)>
    <!ELEMENT URI                    (#PCDATA)>
    <!ELEMENT LangCode               (#PCDATA)>
    <!ELEMENT Text                   (ANY)>
    <!ELEMENT Action                 (execute|save|edit)>
    <!ELEMENT Type                   (#PCDATA)>
    <!ELEMENT Size                   (#PCDATA)>
    <!ELEMENT ControlData            (#PCDATA)>
    <!ELEMENT ControlTarget          (#PCDATA)>
    <!ELEMENT ActionFlag             (#PCDATA)>
    <!ELEMENT ActionID               (#PCDATA)>
    <!ELEMENT CarrierTypeFormat      (#PCDATA)>
    <!ELEMENT CarrierType            (#PCDATA)>
    <!ELEMENT CarrierData            (#PCDATA)>
    <!ELEMENT CarrierPowerState      (#PCDATA)>
    <!ELEMENT CarrierDataReference   (#PCDATA)>
    <!ELEMENT SequenceControl        (CheckExitCode,HaltOnError)>
    <!ELEMENT CheckExitCode          (true|false)>
    <!ELEMENT HaltOnError            (true|false)>
    <!ATTLIST Payload
      encoding                      (plain|base64|xml) "plain"
    >
    <!ATTLIST CarrierData
      encoding                      (plain|base64|xml) "plain"
    >
  ]>
```

As the document type definition implies, the NFC Wrapper does not check whether the NDEF message is formatted correctly according to the NDEF specification. It just converts the NDEF message and all nested NDEF records to XML. A valid Smart Poster record for instance must contain exactly one URI record. The NFC Wrapper doesn't care about the actual number of URI records included in a Smart Poster record. It just returns all present records (which may even be none). It is the responsibility of the higher layer functions to ensure, that all NDEF messages are properly formatted.