FORUM NOKIA

# Nokia 6131 NFC SDK: User's Guide

Version 1.1; July 3, 2007

NFC

NOKIA

Copyright © 2007 Nokia Corporation. All rights reserved.

Nokia and Forum Nokia are registered trademarks of Nokia Corporation. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. Other product and company names mentioned herein may be trademarks or trade names of their respective owners.

### Disclaimer

The information in this document is provided “as is,” with no warranties whatsoever, including any warranty of merchantability, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. This document is provided for informational purposes only.

Nokia Corporation disclaims all liability, including liability for infringement of any proprietary rights, relating to implementation of information presented in this document. Nokia Corporation does not warrant or represent that such use will not infringe such rights.

Nokia Corporation retains the right to make changes to this specification at any time, without notice.

### License

A license is hereby granted to download and print a copy of this specification for personal use only. No other license to any other intellectual property rights is granted herein

# Contents

## Change history

| | | |
|---|---|---|
| March 27, 2007 | Version 1.0 | Initial document release |
| July 3, 2007 | Version 1.1 | Document updated for Nokia 6131 NFC SDK 1.1 |

# 1 Introduction

This document describes how to configure and use the Software Development Kit for Nokia 6131 NFC (Nokia 6131 NFC SDK) to emulate Java™ applications (MIDlets) on corresponding developer platforms. The Nokia 6131 NFC SDK includes the Prototype 4.0 Series 40 MIDP Emulator (resolution 240x320) with corresponding APIs (see Section 1.2, "Supported APIs and configurations"). This document describes the use of the Nokia 6131 NFC SDK in the Microsoft Windows operating system.

Nokia Connectivity Framework (NCF) is integrated with the Nokia 6131 NFC SDK. NCF offers an easy and fast communication environment for the SDKs. Communication between NCF-compatible products can be established via several communication technologies. NCF enables products to communicate with each other with several content types. NCF takes care that the communication and content handling will be easy for the product user.

The Nokia 6131 NFC SDK can be used as a stand-alone kit from the command line or with supported integrated development environment (IDE) tools. The SDK offers MIDP 2.0 support, security domain emulation, and Bluetooth emulation including support for connect-anytime services, mobile 3D graphics emulation, and NFC features emulation. It also provides support for the Unified Emulator Interface (UEI).

The Nokia 6131 NFC SDK installation package includes a Nokia SDK plug-in that enables SDK use and MIDlet development in Eclipse. You can install the plug-in with the Nokia 6131 NFC SDK installation if you want to use the SDK with Eclipse.

## 1.1 Supported development environments

- Microsoft Windows XP (SP2)
- Nokia Connectivity Framework 1.2
- Carbide.j 1.5
- Eclipse 3.1.1 and 3.2
- Sun J2ME Wireless Toolkit 2.2 or later
- Command-line development tools

| **Note:** | Carbide.j 1.5 is not compatible with Eclipse 3.2. |
|---|---|

## 1.2 Supported APIs and configurations

The Software Development Kit for Nokia 6131 NFC supports the following APIs and configurations:

- CLDC 1.1 (JSR-139)
- MIDP 2.0 (JSR-118)
- Nokia UI API 1.1
- Wireless Messaging API 1.1 (JSR-120)
- Wireless Messaging API 2.0 (JSR-205)
- Mobile Media API 1.1 (JSR-135)
- Bluetooth API (JSR-82) excluding OBEX
- Bluetooth API (JSR-82) including Push support and IrDA OBEX (only compiling is possible for IrDA OBEX)
- Mobile 3D Graphics API for J2ME™ 1.1 (JSR-184)
- Java Technology for the Wireless Industry (JSR-185)

- FileConnection Optional Package of PDA Optional Packages for the J2ME™ Platform (JSR-75)
- PIM Optional Package of PDA Optional Packages for the J2ME™ Platform (JSR-75)
- Scalable 2D Vector Graphics API for J2ME™ (JSR-226)
- Web Services JAXP API 1.0 (JSR 172)
- Contactless Communication API (JSR-257)

## 1.3 Supported external card readers

The Nokia 6131 NFC SDK supports the following external card readers:

- OMNIKEY CardMan 5121 (read&write tags, access secure element).
- OMNIKEY CardMan 5321 (read&write tags, access secure element).
- NXP MF RD701 (read&write tags).

For more information about Omnikey CardMan, refer to the Omnikey product pages [9]

For more information about NXP MF RD701, refer to the NXP product pages [7].

## 1.4 Supported tag/card technologies

The Nokia 6131 NFC device supports the following tag/card technologies as a reader/writer:

- MIFARE®: Standard 1k, Standard 4k
- Mifare Ultralight / Type 2 tag
- Mifare DESFire / Type 4 tag
- Sony FeliCa (non-secure parts) / Type 3 tag
- Innovision Topaz and Jewel (read only) / Type 1 tag
- Cards based on ISO 14443-4 (with or without ISO 7816-4) / Type 4 tag
- NFCIP-1 Initiator

The Nokia 6131 NFC device provides the following card emulations / target modes:

- ISO 14443-4A/ISO7816-4 Smart Card (Global Platform-based Java Smart Card)
- Mifare Standard 4k
- NFCIP-1 Target

The Nokia 6131 NFC SDK is capable of emulating the following tag technologies:

- Mifare Standard
- Mifare Ultralight
- Cards based on ISO 14443-4/ISO 7816-4

## 1.5 Setting up the Software Development Kit for Nokia 6131 NFC

To install the Nokia 6131 NFC SDK, run the installation software. For more information on installing the product, see the *Nokia 6131 NFC SDK: Installation Guide* [5].

Information on how to set up the Nokia 6131 NFC SDK with Eclipse is available in Chapter 10, “Using the Nokia 6131 NFC SDK with Eclipse.”

# 2 Installing an external card reader

External card readers require that Windows drivers and possibly other supporting software have been installed before they can be used with the Nokia 6131 NFC SDK.

Install the NXP MF RD701 reader by following the instructions in Section 2.1.1. NXP MF RD701 drivers are in the Nokia 6131 NFC SDK/bin folder.

## 2.1 Installing an Omnikey CardMan reader

Omnikey CardMan reader requires a two-step installation procedure before it can be used with the Nokia 6131 NFC SDK. If these steps are not followed exactly as described here, the card reader might not be able to communicate with the Nokia 6131 NFC SDK. The needed steps are:

- Installing Windows PC/SC driver
- Installing the CardMan Synchronous API

### 2.1.1 Installing Windows PC/SC driver

The drivers for OMNIKey CardMan can be downloaded from the support section of the OMNIKey Web site [8]. Once you have downloaded the drivers, run the installation package. This will extract the driver files into a directory on your hard drive. Take a note of the installation directory, as you will need to enter it later in the installation process.

When an Omnikey CardMan reader is attached to the computer for the first time, Windows starts the driver installation procedure. When a CardMan reader is attached to the computer's USB port, Windows recognizes the new hardware and the dialog shown in Figure 1 is displayed.


Figure 1: Found New Hardware Wizard

**Note:** It is important to select the last item in this dialog! If Windows searches the driver from its own databases, it finds a general smart card reader driver, which does not work with the Nokia 6131 NFC SDK.

Clicking the **Next** button will display the dialog shown in Figure 2.

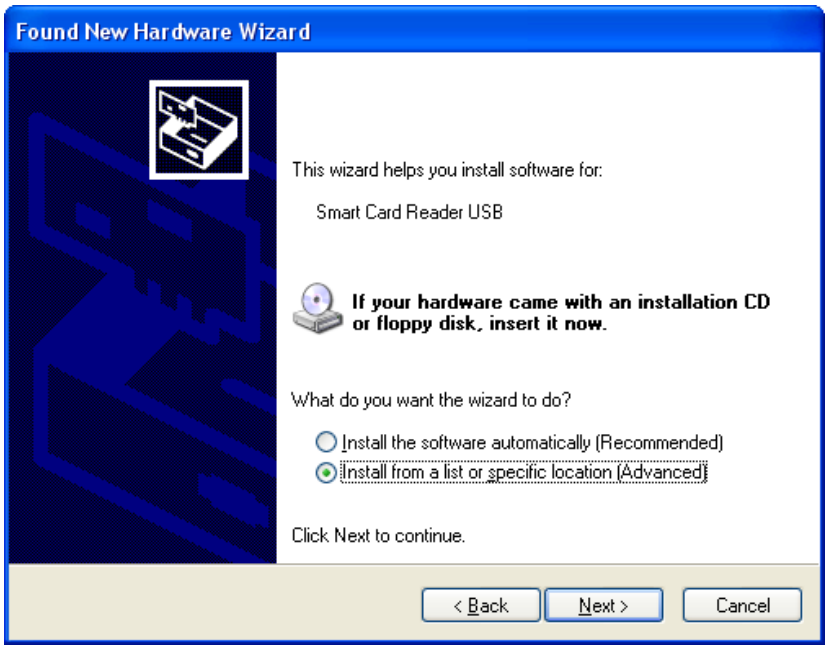


Figure 2: Found Smart Card Reader

Again, the last item in the list should be selected so that Windows allows selecting the driver to install. Clicking the **Next** button displays the dialog shown in Figure 3.

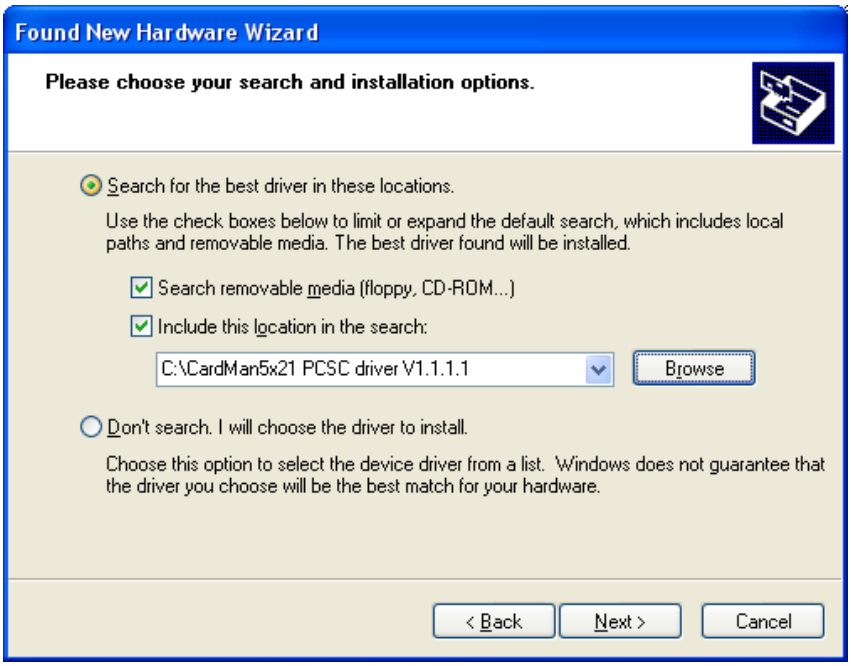


Figure 3: Choose driver installation options

It is easiest to select the driver location in the dialog. Note that the driver name must be “CardMan5x21 PCSC Driver”. There might be other drivers available but the PCSC driver is the one to install. Use the Browse button to select the directory where the driver files were extracted.

When the installation succeeds, Windows displays the dialog shown in Figure 4.

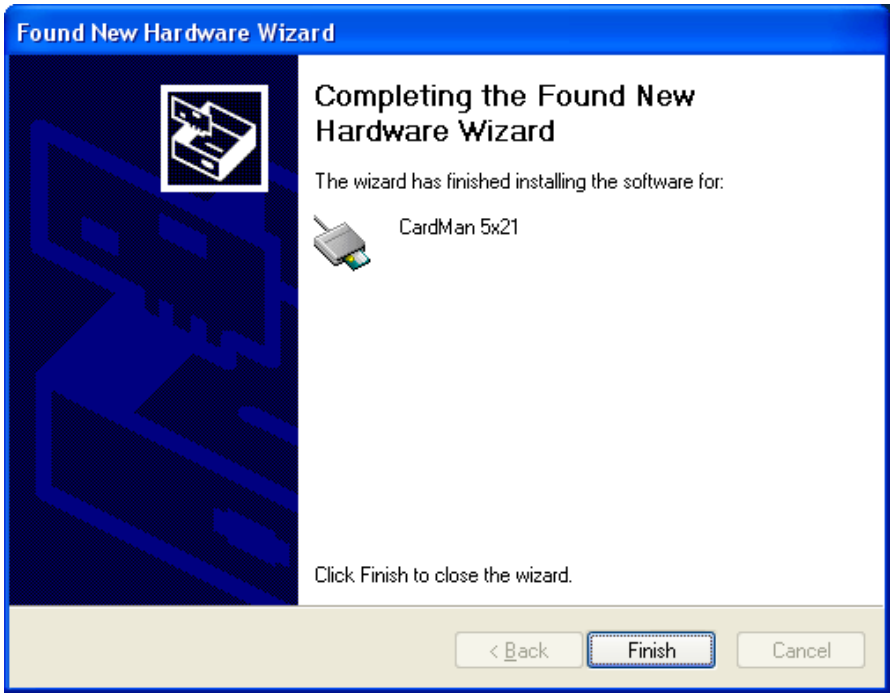


Figure 4: Installation has finished

If the installation was successful, the dialog says “CardMan 5x21” as the device name. If the device name is something else, the driver must be uninstalled and then reinstalled with the correct driver.

### 2.1.2 Installing the CardMan Synchronous API

After the PC/SC driver has been successfully installed, the Synchronous API must be separately installed. The Synchronous API offers the Nokia 6131 NFC SDK a way to communicate with the CardMan external reader.

The installation is a straightforward process with one exception. When selecting the components to install, only DLL files are needed for the Nokia 6131 NFC SDK. See Figure 5.

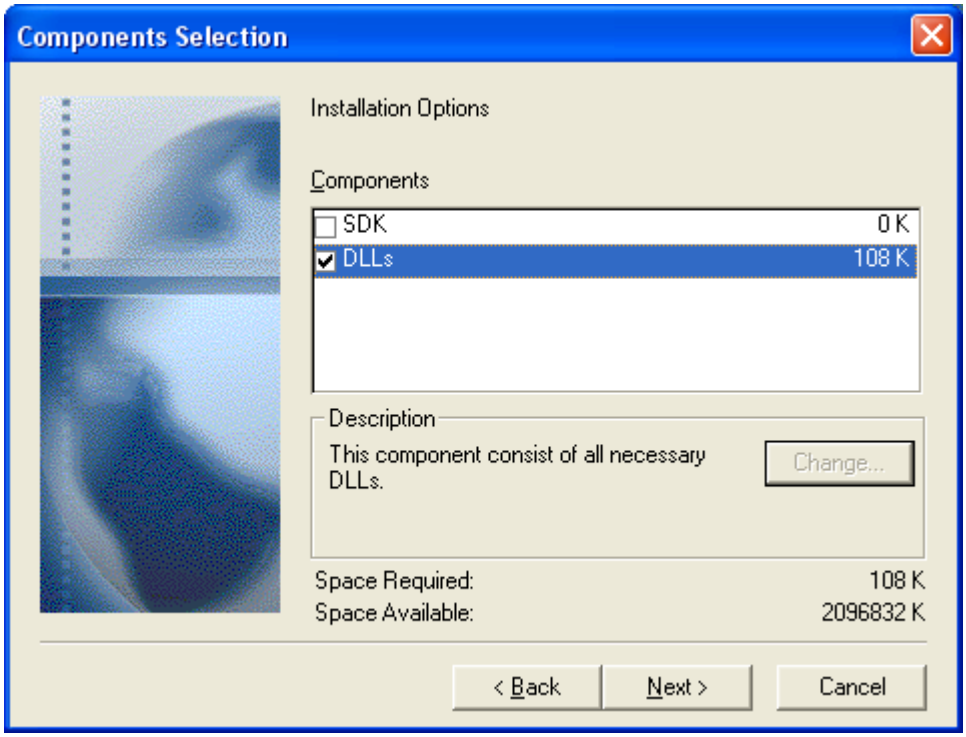


Figure 5: Installation options

### 2.1.3 Verifying CardMan installation

Use the Diagnostic Tool for Cardman in the Windows Control Panel to verify that the CardMan external reader is functioning properly, See Figure 6.

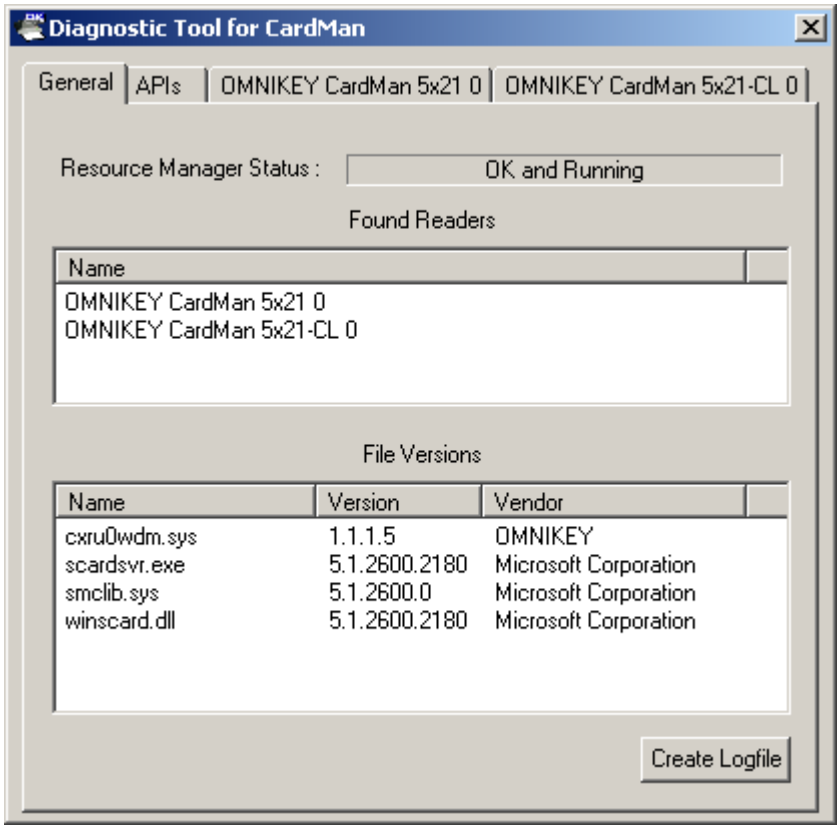


Figure 6: Diagnostic Tool for CardMan

# 3 Using simulated cards

The Nokia 6131 NFC SDK includes simulation of RFID cards. Simulated cards are:

- Mifare Ultralight
- Mifare 1k
- Mifare 4k

Card simulation view is shown on the right side of the emulator as in Figure 7.


Figure 7: Simulated cards utilities

In Figure 7, there are three simulated tags in the Virtual Tags pane. They are all disconnected, The internal tag described in Section 3.1, “Internal Mifare 4, “ is also visible in this pane.

To use a simulated card, drag and drop or double-click it from the dialog to the emulator's “screen” when the emulator is waiting for a card. (If you double-click the tag again, tag status will change to

not connected.) For example, to read and write a simulated tag with the SimpleNDEFExample MIDlet (provided with the SDK), do the following:

1. Import the SimpleNDEFExample MIDlet to Eclipse. Run it with the NFC emulator.
2. Start the SimpleNDEFExample MIDlet in the emulator (so far it was just visible in a list in the emulator).


Figure 8: SimpleNDEFExample MIDlet: select an action to perform

3. Select card writing mode by choosing **Write**. Then click **Select** and **OK**.
4. The MIDlet will show a screen asking for a contact name. This is the data that will be written to the card. You can edit the text or accept the default text. Then click **Accept text**.
5. Depending on the security settings, the emulator might show an alert and ask if the MIDlet can listen to contactless events. Allow the MIDlet to listen.
6. The emulator is now waiting for a card. Select one of the virtual tags and either double-click it or drag it to the emulator screen to activate it.
7. The MIDlet should now show a log of the writing process as shown in Figure 9.

8. Click **Back** to return to previous menu.


Figure 9: SimpleNDEFExample MIDlet: writing to card

The contact name is now written to the virtual tag in the emulator's memory. You can read it by using the **Read** action of the SimpleNDEFMidlet.

Notice, however, that now the simulated card is connected to the emulator: the emulator's phone number is visible below the card name.

## 3.1 Internal Mifare 4k

The internal Mifare 4k represents the internal tag of the Nokia 6131 NFC device, and because of that, it is always connected. For the same reason, it cannot be deleted either. The internal Mifare 4k can be configured to use a simulated Mifare 4k card or a real Mifare 4k tag in an external card reader. To change the properties of the internal card, right click on the Internal Mifare 4k item and select "Edit card" from the pop-up menu. In the opened dialog, you can choose whether the internal card is simulated or placed on an external reader. See Chapter 4, "Using an external card reader," for more information on using an external reader.

# 4 Using an external card reader

After an external reader has been properly installed, the Nokia 6131 NFC SDK emulator automatically detects it. The emulator does not detect attaching or detaching an external reader when running. This means that the external reader must be attached to the computer before the emulator is started.

An application running in the emulator can access a card in the external reader through the JSR-257, Contactless Communication API.

When you want to access an RFID card, simply place the RFID card over the connected external reader. The external reader can access the card from a few centimeters from the reader: the card does not need to physically touch the reader.

If the emulator is running, it sees the card when it is placed over the reader. The application in the emulator can already be running and waiting for a card when the card is placed over the reader. When the card is placed, the application receives an event that a card is now present.

**Note:** The Omnikey CardMan 5x21 reader supports only one RFID card at a time. If two or more cards are placed over the reader, the reader does not detect any of those cards.

**Note:** If either the internal tag or the internal secure card are configured to use a card in the external reader, then the applications do not get events from detected cards in the external reader. In this case, a card placed in the reader acts as if it were the internal card of the Nokia 6131 NFC device.

**Note:** Nokia 6131 NFC has an internal Mifare 4k tag and a secure element. Using an Omnikey Cardman reader, the SDK can access the secure element of the Nokia 6131 NFC. Using an NXP MF RD701 reader, the SDK can access the Mifare 4k tag of Nokia 6131 NFC. If the secure element or tag is not configured to the external reader, placing the Nokia 6131 NFC on the reader results in the application getting an event about the detected Mifare 4k tag or secure element.

## 4.1 Using multiple card readers

Multiple card readers can be attached and installed to the computer where the Nokia 6131 NFC SDK is used. However, the Nokia 6131 NFC SDK can use only one reader at a time. When the emulator is started, it reserves the first reader it finds. The first reader is usually the reader that is first installed to the computer.

If you want to use another reader, you must disconnect the other readers first. All emulator instances must also be closed so that there are no emulators possibly reserving the reader.

When an attached card reader (reserved by the emulator) is detached from the computer, the emulator starts finding other readers. If there are other compatible readers attached to the computer, the emulator again finds the first compatible reader and reserves it.

This allows switching between two or more card readers by attaching and detaching them.

## 4.2 Using the same card reader with multiple emulator instances

As explained in Section 4.1, “Using multiple card readers,” the emulator reserves the first reader it finds at startup. When one emulator instance has reserved the reader, other emulator instances do not see the reader.

There is no possibility to switch the card reader between instances. If you want to use the card reader with another emulator instance, you must close the emulator instance currently reserving the card reader. Then the card reader is available to other emulator instances, and the second running emulator instance can reserve the card reader.

# 5 Developing applications using the secure element of Nokia 6131 NFC

This chapter provides information to developers interested in creating applications using the Nokia 6131 NFC secure element.

The following sections describe how to change the secret device-specific keys of the secure element located in the Nokia 6131 NFC mobile device. The keys need to be changed to manage the applets, Java smart card applications, on the secure element. Additionally, a few hints for applet development and deployment tools are listed. Also practical instructions on how to test the applet on the secure element together with a MIDlet running on the emulator are included.

**Note:** Only MIDlets on the trusted 3rd party security domain can access the secure element of the Nokia 6131 NFC.

## 5.1 Nokia Unlock Service MIDlet

When you purchase a Nokia 6131 NFC mobile device, the secure element contains only secret, device-specific keys for managing the secure element. Nokia Unlock Service MIDlet provides default authentication keys for installing or removing any smart card applications, applets, to/from the secure element of the Nokia 6131 NFC.

After this permanent operation, the secure element on the device may no longer be considered secure by third-party application issuers. Consequently, the device may no longer be used for installing and running third party services that require a secure and trusted connection between the service provider and the secure element, such as payment cards and transport applications.

**Warning:** The secure element cannot be unlocked once it has been locked.

**Warning:** To avoid permanently locking the secure element card manager, do not try to authenticate over 10 times using incorrect keys. If the secure element card manager is locked permanently, you can use the applets in the secure element, but you cannot remove any existing or install any new applets to the locked secure element.

Additionally, the Unlock Service MIDlet changes the Nokia 6131 NFC internal Mifare 4k keys from Nokia private to Mifare default authentication keys.

For more information about the Nokia Unlock Service MIDlet and its availability, refer to http://wiki.forum.nokia.com/index.php/Nokia_6131_NFC_-_FAQs.

Do not delete the following Application IDs from the secure element:

- D276000005AB0503E0040101
- A0000000035350
- D276000005AA040360010410
- D276000005AA0503E00401

The following AIDs are not mandatory for unlocked secure element operation, and may be deleted if required for saving EEPROM space:

- D276000005AA0503E0050101
- D276000005AA0503E00501

### 5.1.1 Key values in the unlocked secure element

The unlock service adds publicly known authentication keys to the secure element. After the unlock operation has been successfully performed, the keyset 42 contains the ENC, MAC, and KEK keys with the hexadecimal value 404142434445464748494a4b4c4d4e4f.

Accessing the device requires ENC+MAC level authentication as specified in the GlobalPlatform specification 2.1.1 secure channel protocol 02 (GlobalPlatform SCP02).

**Note:** After the unlock operation, the secure element keyset 1 still contains the secure authentication keys only known by the trusted 3rd party that performed the device unlock.

The unlocked Mifare 4k card emulation contains the NXP default hexadecimal value FFFFFFFFFFFF for both keys A and B.

## 5.2 Applet development and deployment tools

The following tools can be used for applet development and deployment:

- Sun Java Card Development Kit 2.2.2 [10]
- Giesecke & Devrient, Sm@rtCafé® Professional Toolkit 2.0 [2]
- IBM, JCOP Tools 3.0 [3]

Applets are installed to the Nokia 6131 NFC's secure element with these tools. Refer to documentation of the tools for more information.

The SDK provides three ways to test MIDlets that communicate with Java Card applets: default response ACK 90 00 (OK), connecting to the secure element of an actual Nokia 6131 NFC device, or simulating a smart card. The last two options are described in the following sections.

In the following sections, the secure element is also referred to as smart card or secure card.

### 5.2.1 Mapping a smart card in a card reader as the emulator's internal secure card

The emulator supports mapping a real Nokia 6131 NFC device as the emulator's internal secure card. In practice this means that you can run (or debug) a MIDlet in the emulator that communicates with the secure card in the Nokia 6131 NFC device. In order to utilize this feature, an external smart card reader is needed. Currently the supported readers for this feature are Omnikey CardMan 5121 and 5321.

To map a real Nokia 6131 NFC device's secure card to the internal secure card of the emulator:

1. Install the driver for your reader. Refer to Chapter 4, "Using an external card reader."
2. Connect the reader.
3. Install the applet to the Nokia 6131 NFC device following the instructions of your applet development toolkit (see Section 5.2, "Applet development and deployment tools ").
4. Start the emulator.
5. From the virtual smart cards window, right-click on the Internal Secure Card and select Edit.
6. Select "External Omnikey Reader" in the Secure Card Configuration dialog:

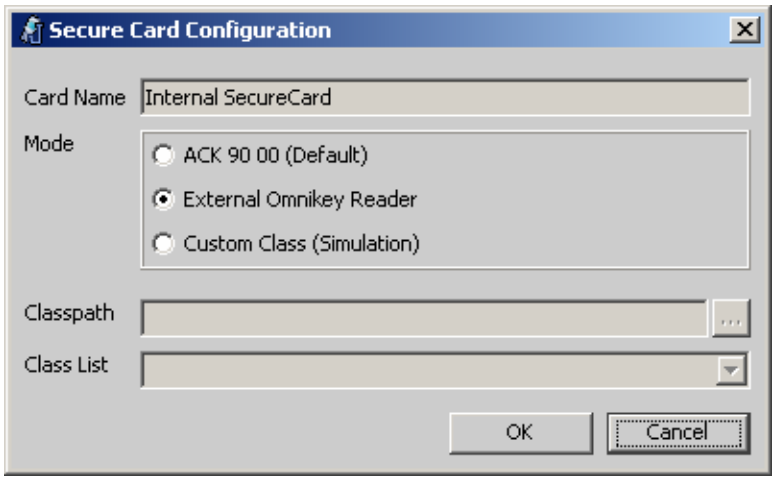


Figure 10: Secure card configuration

After this, the MIDlets running in the emulator can connect to it when you place the Nokia 6131 NFC device on the reader, just like they were running on the actual device .

### 5.2.2 Simulating an internal smart card

The SDK provides also an alternative way to test MIDlets with smart cards. The user can write a class that simulates a smart card and map this class to the internal secure card of the emulator. This is handy, for example, when testing how the MIDlet recovers from erroneous smart card responses.

To map a simulated smart card to the internal secure card of the emulator:

1. Create a simulated smart card class. See the TicketingExampleCards in the SDK for an example of how to do this.
2. Start the emulator.
3. From the virtual smart cards window, right-click on the Internal SecureCard and choose Edit.
4. Choose “Custom Class (Simulation)” in the Secure Card Configuration dialog:

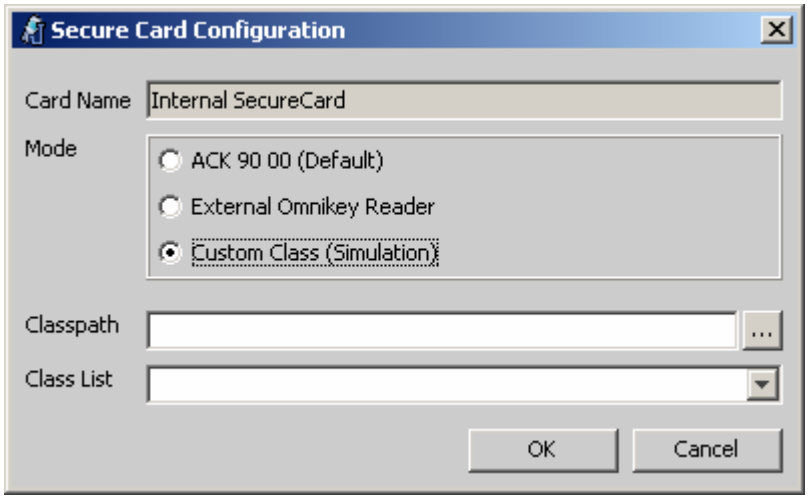


Figure 11: Choosing custom class

5. In the Classpath field, select the ZIP or JAR file containing your class.

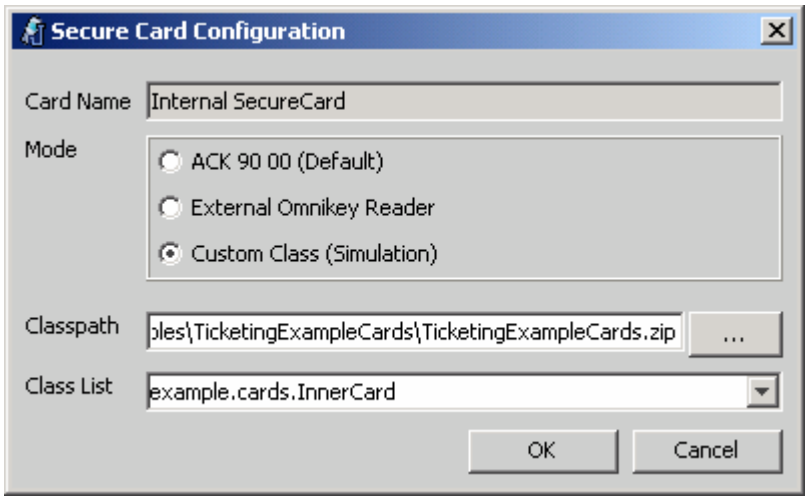


Figure 12: Setting custom class

6. In the Class List field, select the class you want to test.

After these steps, calls to the internal secure card are routed to the class that was selected in the configuration dialog.

# 6 Simulating an external smart card

The SDK also provides a way to simulate external smart cards. The user can write a class that simulates an external smart card and map this class to a virtual smart card in the emulator. This is handy, for example, when testing how the MIDlet recovers from erroneous smart card responses.

To create and map a simulated external smart card to a virtual external smart card of the emulator:

1. Create a simulated external smart card class. See the TicketingExampleCards in the SDK for an example of how to do this.
2. Start the emulator.
3. From the virtual smart cards window, right-click on the panel and select “Create”.
4. Enter a name for the new simulated external smart card.
5. Choose “Custom Class (Simulation)” in the Create a New Secure Card Configuration dialog:

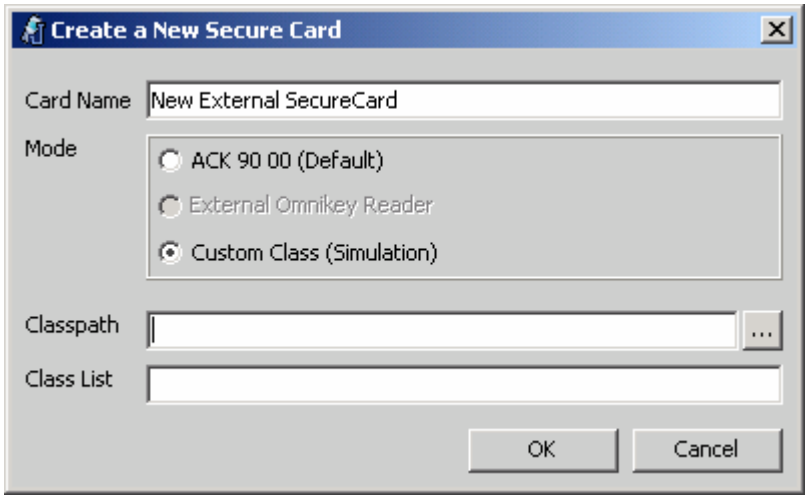


Figure 13: Choosing custom class

6. In the Classpath field, select the ZIP or JAR file containing your class.

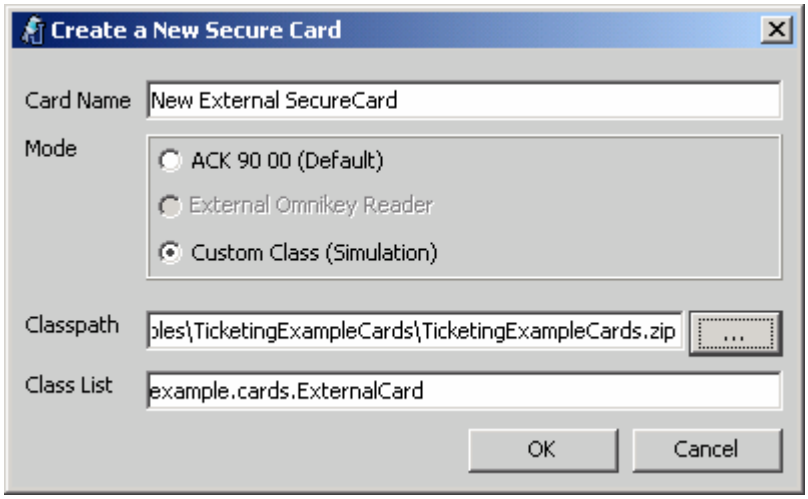


Figure 14: Setting custom class

7. In the Class List field, select the class you want to test.

After these steps, a new simulated external smart card will be available in the virtual smart cards window. To use the new smart card, every time a MIDlet is waiting for a connection to a smart card, you can attach the simulated one by double-clicking on it or by dragging it to the emulator screen.


Figure 15: New simulated smart card attached to the emulator when using the TicketingExample

# 7 Using the Nokia 6131 NFC SDK from Windows Start menu

To start the MIDlet emulator, select Launch Emulator from the Start Menu. The program opens a small window as shown below:


Figure 16: Nokia SDK Launcher

To start a MIDlet, choose Open... from the File menu. You will be prompted to select a JAD file.

The purpose of this example is to launch the SimpleNDEFExample included in the SDK by choosing the JAD file (the path may be different if you didn't install the SDK in the default location):
`C:\Nokia\Devices\Nokia_6131_NFC_SDK_1_1\examples\SimpleNDEFExample\SimpleNDEFExample.jad`

Once you have selected a JAD file, the MIDlet emulator will be launched. The emulator will show the Applications menu which lists the MIDlet you have chosen to run. Now you can run the MIDlet using the emulated phone.


Figure 17: Emulator in Applications menu

# 8 Using the Nokia 6131 NFC SDK with Nokia Carbide.j

The Nokia 6131 NFC SDK can be used with Nokia Carbide.j. This chapter describes how to install and configure the SDK and start to develop applications.

## 8.1 Installation and configuration

You can install and configure the Nokia 6131 NFC SDK emulator to work with Nokia Carbide.j as follows:

1. Install the Nokia 6131 NFC SDK to your local file system (see the Installation Guide [5]).
2. Start Nokia Carbide.j.
3. Select **Emulators** | **Configure Emulators**. The **Configure Emulators** dialog opens. (See Figure 18.)
4. If the Prototype SDK does not appear in the **Emulators** list, click the **Add...** button.
5. Browse to the UEI emulator root directory in the file system and click **Open**. Alternatively, browse to the emulator configuration file, `<Product_name>.xml`, under the SDK installation directory and click **Open**.

The SDK is added to Nokia Carbide.j and is displayed in the **Emulators** list. Devices of different series are displayed in the **Select default device** drop-down menu. You can set any SDK as the default emulator by selecting it from the **Emulators** list and clicking the **Set as Default** button. Then select the emulator used from the **Select default device** drop-down menu. (See Figure 18)

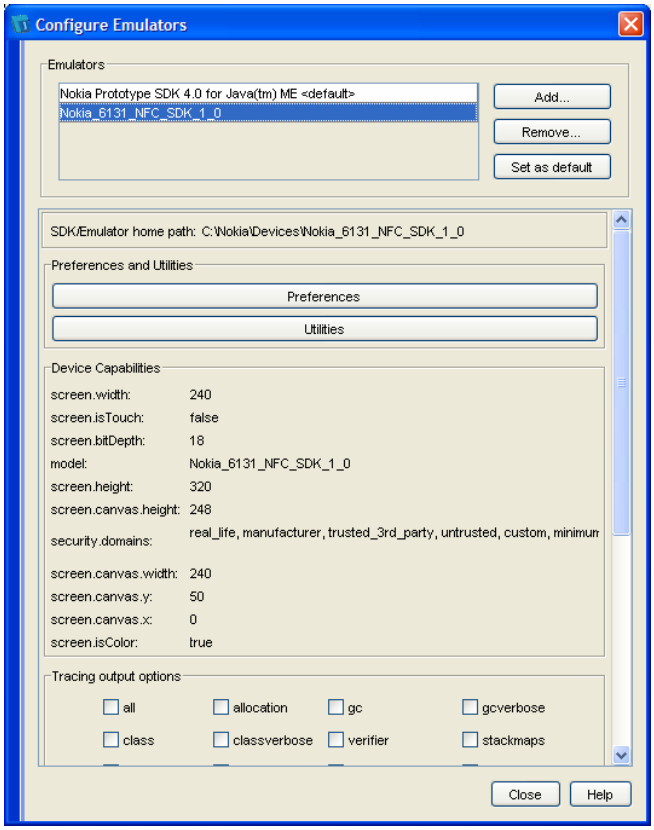


Figure 18: Configure Emulators dialog

To launch the SDK's preferences dialog, click the **Preferences** button, select the device, and click **OK**. For more information, see Chapters 13, "Emulation settings," and 14, "Emulating MIDlet suite security issues," that cover performance restrictions and security settings.

To remove the Nokia 6131 NFC SDK from Nokia Carbide.j:

1. Open the **Configure Emulators** dialog by selecting **Emulators** | **Configure Emulators...**
2. Select the product you want to remove from the **Emulators** list.
3. Click the **Remove...** button.

Note that this only removes the SDK from Nokia Carbide.j. The SDK must be uninstalled separately.

## 8.2 Development

When you have successfully installed and configured the Nokia 6131 NFC SDK, you can start developing MIDlets and running them on the emulators:

1. Create or open a MIDlet project as defined in the Nokia Carbide.j User's Guide [1].
2. Select **Emulators** | **Start Emulators....**
3. Select an SDK from the **Emulators** list.
4. Click the **Emulate** button to start the application.

# 9 Using the Nokia 6131 NFC SDK with Nokia Connectivity Framework

Nokia Connectivity Framework (NCF) covers the core functionalities of communication. It manages communication between products in a local machine and with real devices. Nokia Connectivity Framework consists of different technology and communication components.

## 9.1 Installation and configuration

Nokia Connectivity Framework is integrated with the Nokia 6131 NFC SDK. NCF is installed in the local workstation with the Nokia 6131 NFC SDK installation. NCF, which has been installed in the local machine, is updated with new components when a new product is installed in the local machine.

To configure the product properties of an emulator, modify the product integration `.xml` files from `<NCF Home Directory> \integrations \<Product_Name>`.

For example:

- To change the product's home directory that sets the value for the environment variable `_PRODUCT_HOME_`, modify `<ProductName>.xml`.
- To change the SMS Inbox, modify `<ProductName>FileHandler.xml`.
- To change the SMS Outbox, modify `<ProductName>FileHandlerServer.xml`.
- To change the VCOM port, modify `<ProductName>VCOMServer.xml`.

You can alternatively use the NCF user interface to modify the product properties. The NCF User's Guide [6] explains how to do that.

**Note:** You need to restart the NCF after configuration for the new properties to take effect.

## 9.2 NCF startup

### 9.2.1 Automatic startup

NCF starts automatically when you log on to the Windows workstation. The startup link for NCF is located in the Windows **Start | Programs | Startup** menu. You can remove the automatic startup by deleting the Nokia Connectivity Framework shortcut from the **Start | Programs | Startup** menu.

When NCF is started, the Nokia Connectivity Framework icon appears on the system tray. If you click the icon and select the **What's This?** option, the information text of NCF appears on the screen.

### 9.2.2 Manual startup

Start the Nokia Connectivity Framework by selecting Windows **Start | Programs | Nokia Developer Tools | Nokia Connectivity Framework | Nokia Connectivity Framework.**

### 9.2.3 Manual startup in debug mode

Start the Nokia Connectivity Framework in debug mode by executing the following startup script: `<NCF Home Directory>\bin\startNCF.cmd`.

### 9.2.4 NCF shutdown

To shut down the NCF service, click the NCF Service icon in the system tray and select **Shutdown**.

# 10 Using the Nokia 6131 NFC SDK with Eclipse

The Nokia 6131 NFC SDK installation package includes the Nokia SDK Plug-in that integrates the SDK with Eclipse. The Nokia SDK Plug-in enables you to run and debug MIDlets in Eclipse. It is recommended that Carbide.j version 1.5 is used. Carbide.j provides tools for mobile Java development.

## 10.1 Integration

You can install the Nokia SDK Plug-in with the Nokia 6131 NFC SDK installation. In the installation wizard, select the **Nokia 6131 NFC SDK Integrated with Eclipse** choice in the **Choose Install Set** window. Select the root directory of Eclipse in the file system. For more information, see the *Nokia 6131 NFC SDK: Installation Guide* [5].

## 10.2 Using the SDK Plug-in with Eclipse

The Nokia SDK Plug-in is an independent plug-in that enables you to run and debug MIDlets in Eclipse. The SDK Plug-in also enables the use of Personal Profile applications but they are not used with the Nokia 6131 NFC SDK.

**Note:** To be able to use this feature, you need to be in the Java Perspective in Eclipse. The perspective can be opened from **Window | Open Perspective | Java**.

### 10.2.1 Creating a new MIDP project

To create a new MIDP Project:

- Choose **File | New | Project...** in the Java Perspective. Then choose **Java | MIDP Project (Nokia SDK Plug-in)**, and click **Next** to continue.
- Enter a name for the project. If you choose a directory for the project content, make sure the project name is the same as the name of the content directory. Click **Next** to continue.
- Select the SDK you wish to use for compiling and preverifying the MIDP project. If the SDK is not displayed in the drop-down list, click **Browse...** and select the root directory of the SDK (for example, `C:\Nokia\Devices\NFC_SDK`). Click **Next** to continue.
- Make sure that the project source folder on build path and the default output folders are set correctly (for example, `MyProject\src` and `MyProject\bin`) on the **Java Settings | Source** tab. Click **Finish** to continue.

**Note:** Use only one source folder for your MIDlet. Nokia Carbide.j only supports one source folder at a time. Files added to any other folders are not recognized.

### 10.2.2 Changing the SDK used for compilation and preverification

- Select your project in the Package Explorer and choose **Project | Properties**.
- Select SDK Plug-in properties and click **Browse...** Select the SDK root directory (for example, `C:\Nokia\Devices\NFC_SDK`) to add the SDK to the Eclipse IDE as a new Java Runtime Environment (JRE).
- Click **OK** to approve the changes to the project properties.

### 10.2.3 Running a MIDlet

The first time you run a MIDlet you have to create a launch configuration for it. Select your project or class in Package Explorer and select **Run | Run...** from the menu. Choose Nokia SDK Plug-in and click

the **New launch configuration** button. Type a name for the launch configuration and check that the Main class is correctly filled. Press **Run** to start the emulator. Figure 19 shows the run configuration for the SimpleNDEFExample MIDlet.

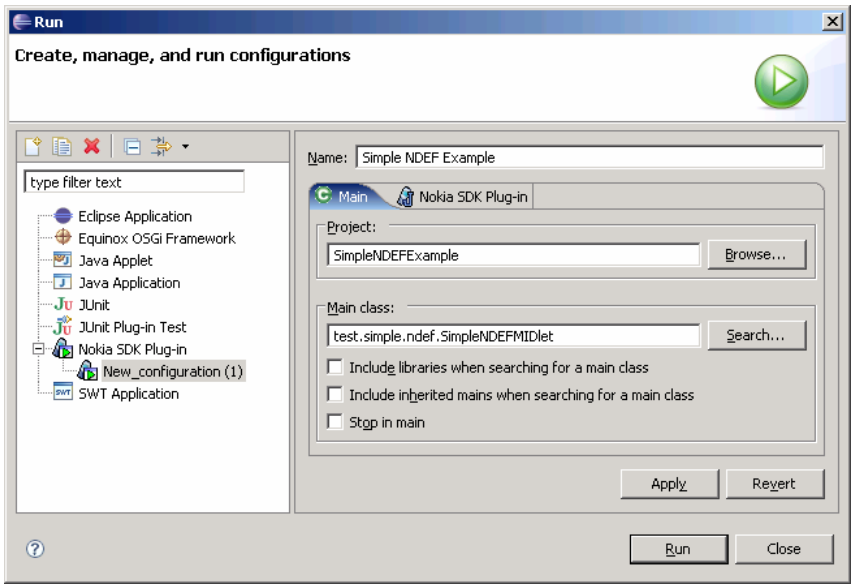


Figure 19: Adding a run configuration in Eclipse

The next time you want to run your MIDlet, you can simply click **Run | Run...**, select the existing launch configuration, and click **Run**. You can also select the launch configuration from the **Run | Run History...** menu.

You can select another emulator for the launch configuration by editing the launch configuration. Choose **Run | Run...** The dialog for creating, managing, and running configurations opens. Select a launch configuration under the Nokia SDK Plug-in in the configurations list. On the Nokia SDK Plug-in tab, you can select which SDK to use and launch SDK-specific dialogs for preferences and utilities. Select the trace options you wish to use by selecting the appropriate check boxes.

### 10.2.4 Debugging a MIDlet

To debug a MIDlet you first need to create a launch configuration as described in the previous section. The same launch configuration can be used for both running and debugging.

Add breakpoints to the MIDlet. Select **Run | Debug...**, run your launch configuration, and finally click **Debug**. The emulator starts and you can debug the MIDlet.

You can also run the launch configuration from the **Run | Debug History...** menu if you have debugged it before.

To modify configuration settings, choose **Run | Debug...** The dialog for creating, managing, and running configurations opens, and you can perform actions similar to the ones described in Section 10.2.3, “Running a MIDlet.”

**Note:** It is recommended that you disable all Suspend Execution options from **Window | Preferences | Java | Debug**. It is also recommended that you increase the **Debugger timeout (ms) setting** to 20,000 when debugging with SDKs (see Figure 20).

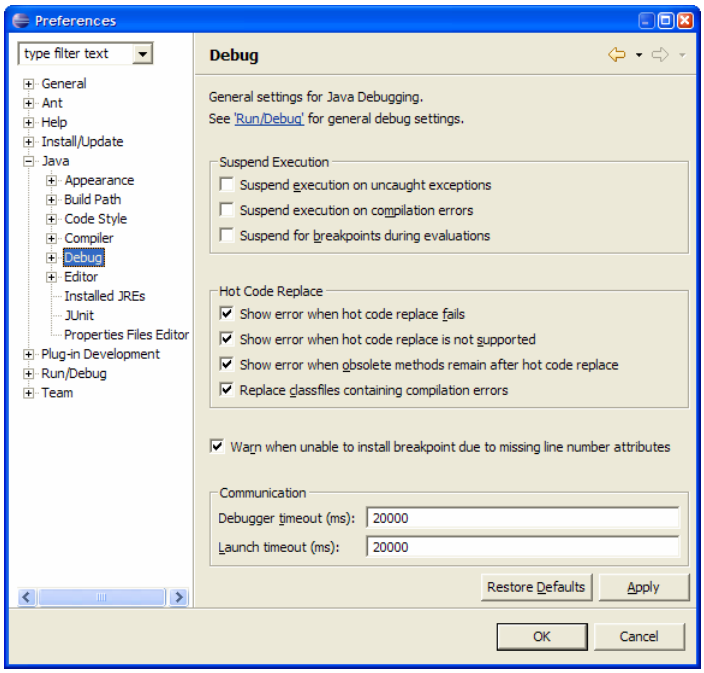


Figure 20: Recommended settings for the Window | Preferences dialog

# 11 Using the Nokia 6131 NFC SDK with Sun Java Wireless Toolkit (WTK)

## 11.1 Integration

The Nokia 6131 NFC SDK appears in the Wireless Toolkit's device list if it has been installed in the `<Java WTK home dir>\wtklib\devices` directory.

See the WTK User's Guide [4] for help in using WTK.

**Note:** If you have installed the Nokia 6131 NFC SDK in a location different from the one above, you can copy the entire SDK directory to the `devices` directory, but you should then modify the Nokia Connectivity Framework integration files of each device as described in Section 9.1, "Installation and configuration."

## 11.2 Development

When using the Java Wireless Toolkit and the Nokia 6131 NFC SDK, the following issues affect the building and running procedures of the application:

- The build process does not automatically refresh the MIDlet JAR package after building the application. Choose **Project | Package | Create Package** from the menu to refresh the JAR package before running the application.
- About the **Preferences** window: The clean database utility of the Java Wireless Toolkit has no effect on the Nokia 6131 NFC SDK. The only way to clean up the RMS database of the SDK is to manually delete the application-generated content of the `appdb` directory.

# 12 Using the Nokia 6131 NFC SDK from the command line

The Nokia 6131 NFC SDK can also be used from the command line. The command line access uses the Unified Emulator Interface (UEI) syntax.

## 12.1 UEI command-line syntax

```
emulator [arguments] <Application>
```

| Argument | Description |
|---|---|
| -classpath, -cp | The class path for the VM. |
| -D<property=value> | Property definitions. |
| -version | Displays the Nokia 6131 NFC SDK version information. |
| -help | Displays the list of valid arguments. |
| -Xverbose[: allocation \| gc \| gcverbose \| class \| classverbose \| verifier \| stackmaps \| bytecodes \| methods \| methodsverbose \| frames \| stackchunks \| exceptions \| events \| threading \| monitors \| networking \| all ] | Enables verbose output. |
| -Xquery | Query options. |
| -Xdebug | Enables the remote debugger. |
| -Xrunjdwp:[transport=<transport>, address=<address>,server=<y/n> suspend=<y/n>] | Debugging options. |
| -Xdevice:<device name> | Name of the SDK to be emulated. |
| -Xdescriptor:<jad file name> | The JAD file to execute. |
| -Xjam[:install=<jad file url> \| force \| list \| storageNames \| run=[<storage name> \| <storage number>] \| remove=[<storage name> \| <storage number> \| all] | Java Application Manager and support for over-the-air provisioning (OTA). The Nokia 6131 NFC SDK supports the http and file URLs. (e.g., `file://C:/MyMIDlet.jad`)<br>Note that <jad file url> can include userinfo for authentication purposes. (e.g., `http://user:password@127.0.0.1:8081/MyMIDlet.jad`). |
| -Xheapsize:<size> | Specifies VM heap size (e.g., 65,536 or 128 KB or 1 MB). |

## 12.2 Debugging from the command line

To debug the execution of an application launched from the command line, you need an external debugger. You can use the Java Debugger included in the Java Development Kit by Sun Microsystems.

Basic steps for debugging:

1. Start the SDK in debugging mode. The easiest way to do this is to use `emulator.exe` in the bin directory of the SDK:

```
emulator.exe -Xdebug -Xrunjdwp:transport=dt_socket,address=4452,
server=y -Xdescriptor:D:\WTK22\apps\UIDemo\bin\UIDemo.jad
```

This puts the SDK debugger connection in wait state. The SDK waits for the debugger connection to port 4452.

2. Launch the debugger to connect with the SDK:

```
jdb -connect com.sun.jdi.SocketAttach:hostname=localhost,port=4452
```

The debugger is connected to the SDK's debugger port when you see the text "Connection received" in the SDK's prompt. Type `help` in the debugger prompt to see the available debugger commands. The `run` command starts MIDlet execution.

# 13 Emulation settings

## 13.1 Proxy settings and performance simulation

You can set proxy settings and various restrictions for the emulator in the Nokia 6131 NFC SDK on the **Emulation** tab of the **Preferences** dialog (Figure 21). All performance simulation options are disabled by default. The changes will take effect immediately or after restarting the emulator, depending on the restriction you have changed. The procedure needed for the changes to take effect is detailed in the description of each option.

| Setting | Description |
|---|---|
| Use HTTP proxy (restart emulator) | You can configure the emulator to communicate through a proxy. Enter the proxy address and port number. You need to restart the emulator for this change to take effect. |
| Enable network speed emulation | You can define the maximum write and read throughput (bits/sec) for network traffic by enabling network speed emulation. You can enter the value manually or you can select one of the predefined values from the drop-down list. The values range from 1,200 to 112,000 bits/sec. |
| Enable graphics primitives latency | This value defines the latency between each graphics call. Select one of the predefined values that range from 0 to 40 ms. |
| Enable RecordStore size emulation | Specifies the size of the RecordStore available. The size values range from 10,000 to 100,000 bytes. If the RecordStore size exceeds the specified value, a warning message is displayed on the prompt. |
| Enable memory size emulation (restart emulator) | Specifies the heap size available to the SDK. Available heap sizes range from 32 KB to 64 MB.<br>If the MIDlet exceeds the specified heap size, the following message is displayed on the prompt: *"Your MIDlet has exceeded its warning size threshold of ***** bytes by ***** bytes"* |
| Enable MIDlet JAR size emulation | Specifies the maximum JAR file size that can be used. The predefined values range from 10,000 to 100,000.<br>If the JAR file exceeds the specified value, the following message is displayed on the prompt: "*Your MIDlet JAR size has exceeded its warning size threshold of **** bytes by ****** bytes.*" |

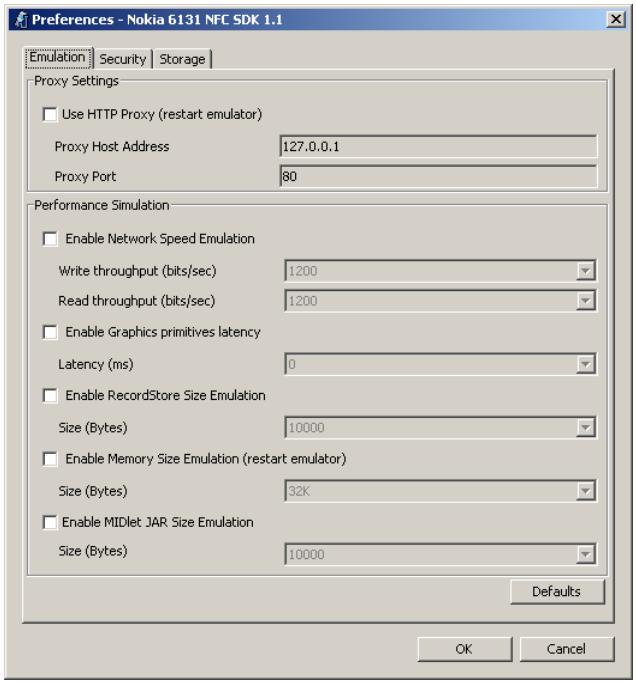


Figure 21: Emulation Preferences dialog

# 14 Emulating MIDlet suite security issues

The Nokia 6131 NFC SDK has the capability to emulate the security levels of MIDlet suites defined in the MIDP 2.0 specification. This includes MIDlet authentication, authorization, and the steps required by the MIDlet to perform protected actions.

Use the **Security** tab (Figure 22) in the **Preferences** dialog to set the Security simulation level for the emulator.

The **Description** field gives a short description of the currently selected security simulation level. The default simulation mode is **Untrusted**.

**Note:** If the emulator is running, you must restart it after changing the settings in order for your new settings to take effect.

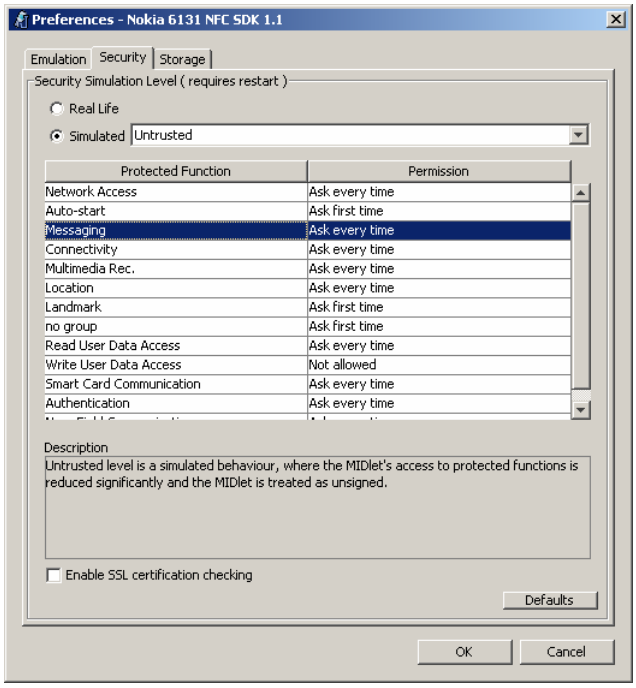


Figure 22: Security Preferences dialog

## 14.1 Security domain settings

In Real Life mode, MIDlets are treated similarly to how a real device would handle them. MIDlets gain access to protected functions based on certificate information. MIDP 1.0 and unsigned MIDlets are always treated as untrusted. If the signing certificate cannot be matched to the root certificate on the SDK or device, the MIDlet is rejected.

**Note:** In Real Life mode you always need a JAD/JAR pair to run a MIDlet. It is not possible to run a MIDlet using a class in this mode.

In addition to the Real Life mode, there are five other simulated security modes:

- Manufacturer level is a simulated level where the MIDlet is treated as a MIDlet created by Nokia. The MIDlet is provided with access to protected functions accordingly.
- Trusted 3rd-Party level is a simulated level where the MIDlet is treated as certified by a trusted third-party.

- Untrusted level is a simulated level where the MIDlet's access to protected functions is reduced significantly and the MIDlet is treated as unsigned.
- Custom level is a simulated level where the user can set the permissions for supported protected functions. When the settings are customized, the SDK will not work like a real device.
- At the minimun security level, the MIDlet does not have access to protected functions.

When the MIDlet is started, it is treated as if it belonged to the selected security domain. The MIDlet is provided with access to protected functions accordingly.

## 14.2 Application permission setting

The end user may also grant access to protected functions to an application after the MIDlet has been run in Trusted 3rd-Party, Untrusted, or Custom simulation mode. You can emulate this by selecting the desired security domain and then setting the needed permission for the MIDlet.

Each security domain contains five application permission settings that can be modified. The settings are: *Network Access*, *Auto-start*, *Messaging*, *Connectivity*, and *Multimedia recording*.

- *Network access* represents access to any function that results in an active network data connection (for example, GSM, GPRS, UMTS).
- *Auto-start* represents access to any function that allows a MIDlet suite to register for a push event.
- *Messaging* represents access to any function that allows sending and receiving messages (for example, SMS).
- *Connectivity* represents access to any function that activates a local port for further connection (for example, COM port, IrDA, Bluetooth).
- *Multimedia recording* represents access to any function that can capture still images or record video or audio clips.

There are four types of permissions: "*Not allowed*," "*Ask every time*," "*Ask first time*," and "*Always allowed*." Note that some of the permission types are not available for application permissions in certain security domains.

You can restore the default values for all security simulation levels by selecting **Restore**.

## 14.3 SSL certification checking

You can enable the SSL certification checking by selecting the appropriate check box. When this option is enabled, the emulator checks that the certificate located on the server where the MIDlet is trying to make a SSL connection has been issued to that particular server.

## 14.4 Requirements for a signed MIDlet suite

Both JAD and JAR file are required for signed MIDlet suites. The signing of a suite is checked in the Nokia 6131 NFC SDK MIDP Emulator only when the Real Life simulation level is used.

A MIDlet suite can be signed using Nokia Carbide.j after the application descriptor and JAR file contain all the required attributes listed below.

The application descriptor (JAD file) MUST contain the following attributes:

- MIDlet-Name
- MIDlet-Version
- MIDlet-Vendor

- MIDlet-Jar-URL
- MIDlet-Jar-Size
- MIDlet permissions for protected API functions (See the Nokia Carbide.j User's Guide [1] for further information.)

The manifest (inside the JAR file) MUST contain the following attributes:

- MIDlet-Name
- MIDlet-Version
- MIDlet-Vendor

The manifest (inside the JAR file) or the application descriptor (JAD file) MUST contain the following attributes:

- MIDlet-<n> for each MIDlet
- MicroEdition-Profile
- MicroEdition-Configuration

For further information about the attributes, see the `javax.microedition.midlet` package found in Javadocs.

The MIDlet suite signing process involves the MIDlet suite and public key certificates. Public key authentication is based on a set of root certificates, which are located in the Nokia 6131 NFC SDK MIDP Emulator. The signer of the MIDlet suite (usually the developer or distributor of the MIDlet suite) needs to have a valid public key certificate that can be validated to one of the root certificates. Public key certificates can be requested from one of the common certificate authorities (CA) like Verisign, Inc. or Thawte. If public key verification fails, the signed MIDlet suite cannot be installed or run.

Inserting certificates into the application descriptor can be done using Nokia Carbide.j. For more information, see the Nokia Carbide.j User's Guide [1].

After signing, the application descriptor should have the following lines:

```
MIDlet-Certificate-<n>-<m>: <base64 encoding of a certificate>
MIDlet-Jar-RSA-SHA1: <base64 encoding of Jar signature>
```

## 14.5 Adding SSL certificates to the SDK keystore

You can add SSL server certificates used in secure SSL connections to the emulator using J2SE's keytool and Sun's MEKeyTool. See below for a brief description on how this can be done. For further information on how to use keytool or MEKeyTool, see the following:

- **Keytool**: http://java.sun.com/j2se/1.3/docs/tooldocs/win32/keytool.html
- **MEKeyTool** (from command line: `java -jar <MEKeyTool_path>\MEKeyTool.jar -help`

To add a certificate to the SDK's keystore:

1. Import a valid certificate to the JDK keystore or your own keystore using keytool (import option). Give an alias to your new certificate.
2. Verify that the keystore you are using contains the new certificate by listing the contents of the keystore with the keytool list option.
3. Import the new certificate from the keystore you are using to a NFC SDK's keystore with the MEKeyTool import option.

4. Verify that the SDK keystore contains your new certificate by listing the contents of the keystore with the MEKeyTool list option.

Importing a certificate to your keystore can be done with the keytool utility from the command line (keytool must be set in PATH):

```
keytool -import -file <cert_file> -alias <alias> [-keystore <keystore>]
[-storepass <storepass>]
```

You can verify from the command line that your certificate has been added to the keystore:

```
keytool -list [-keystore <keystore>] [-storepass <storepass>]
```

New certificates can be imported/listed from the command line with MEKeyTool:

```
cd <emulator installation path>
```

Import certificates:

```
java -jar <MEKeyTool path>\MEKeyTool.jar -import -alias <alias> -
keystore <keystore> -storepass <storepass> -domain trusted –MEkeystore
<Emulator path>\appdb\_main.ks
```

List certificates:

```
java -jar <MEKeyTool path>\MEKeyTool.jar –list –MEkeystore <Emulator
path>\appdb\_main.ks
```

# 15 Included application programming interfaces

The APIs mentioned in this chapter are included in the Nokia 6131 NFC SDK, which is based on Nokia Prototype SDK 4.0.

## 15.1 Mobile Media API (JSR-135)

The Multimedia API provides an interface for audio and multimedia capabilities of a device running Java ME. This small-footprint multimedia API also addresses scalability and support for more sophisticated features in devices with limited memory size and processing capabilities. The API enables you to use the media recording capabilities included in the device. This section explains how these features are emulated in the Nokia 6131 NFC SDK.

### 15.1.1 Camera

You can use the `getSnapshot` method in `javax.microedition.media.control.VideoControl` to get a snapshot. The emulator uses the files located in the `camera` directory in the SDK's installation directory for camera emulation. You can change the files presented in Figure 23 to suit your needs but remember to maintain the aspect size and format of the images.


Figure 23: Viewfinder and snapshot images in supported capture formats

### 15.1.2 Audio recording

You can record live audio with the Nokia 6131 NFC SDK by using a microphone and the Mobile Media API. You should test that your microphone is working properly before using the SDK for audio recording purposes. If you do not have a microphone, you can set *Stereo Mixing* or *Line in* as your audio source. In Windows the recording options are available in the **Recording Control** dialog box.

## 15.2 PDA Optional Packages for the J2ME™ Platform (JSR-75)

The Nokia 6131 NFC SDK includes the packages, classes, and interfaces described in the FileConnection (FC) Optional Package 1.0 and Personal Information Management (PIM) Optional Package 1.0 specifications, as well as a security model for accessing these APIs.

### 15.2.1 FileConnection API

Accessing file systems is a common feature on many PDAs and other mobile devices. The FileConnection API is an optional package that gives Java ME devices access to file systems that reside on mobile devices. Specifically, the API allows access to removable storage devices, such as external memory cards.

The file system of each Nokia 6131 NFC SDK is located under the installation directory (see Figure 24).

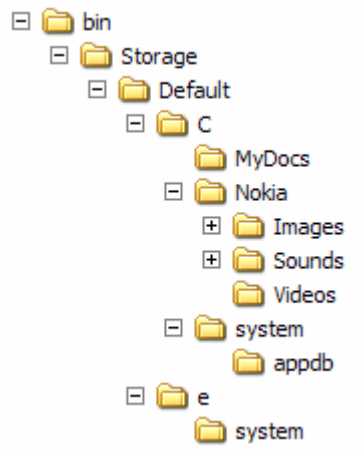


Figure 24: File system structure

When you start an instance of the SDK, it creates a file system for the instance. The instance's file system is a copy of the default file system and its name is the instance's sequence number (see Figure 25). The first instance's file system is placed under the folder '0', and if you start another instance while the first instance is still running, the second instance will have a file system under the folder '1'. If you close and restart instances during run time, in the new startup the first released folder is reserved for the started instance. This kind of a file system enables you to test, for example, an application that communicates with another emulator instance and then saves the received data to the RecordStore.

| **Note:** | You can restore device default settings by deleting the numbered folders. The next time you start an instance, the default folder and its contents are copied again to the started instance. |
|---|---|

You can copy the files you want to test with your MIDlet to the default file system, after which they will be available for every emulator instance that you start.

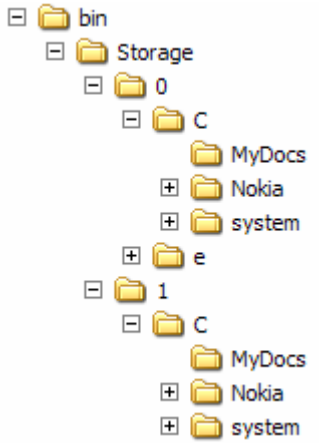


Figure 25: Two emulator instances running simultaneously with separate file systems

The file system can be controlled on the **Storage** tab of the **Preferences** dialog (see Figure 26). The available storages are listed in the **Storage** column and the connection type for each storage is presented in the **Connection** column.

- **Needs reboot** means that the storage can only be connected by turning off the device. For example, you must turn off the device when you want to change a multimedia card located

beneath the device's battery. Select **Connect on start-up** or **Do not connect on start-up** from the **Status** column to control whether the storage should be mounted during startup.

- **Connect while running** means that the storage can be connected while the device is operational. Select **Connected** or **Not connected** from the **Status** column to mount the storage.

You can select the **Capacity** of the memory card(s) and the native file system by selecting a value from the drop-down list. Select **Defaults** to use the default settings. The emulator must be restarted for the changes to take effect.

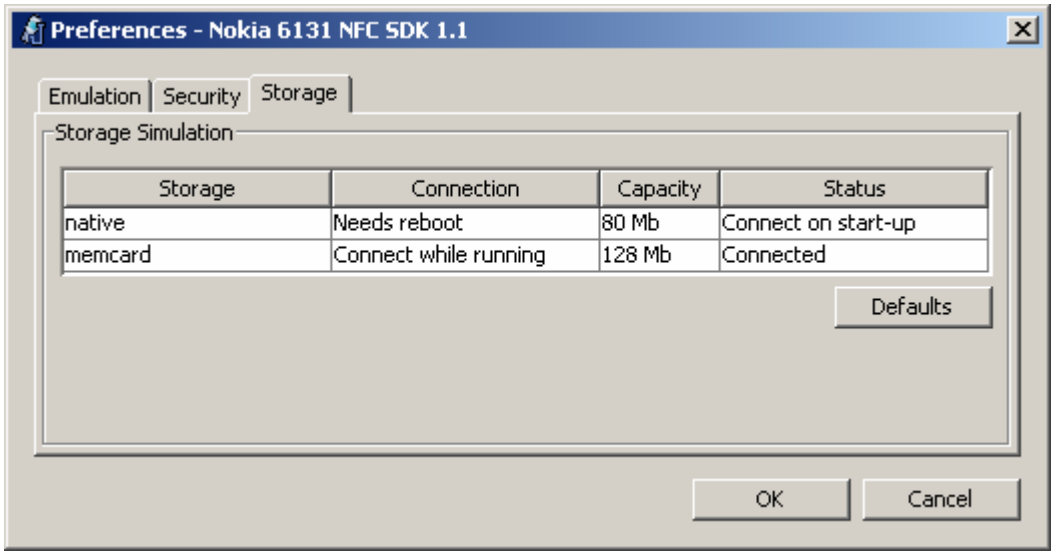


Figure 26: Storage Preferences dialog

The FileConnection API, MIDP 2.0, and the emulated file system set security requirements for the API implementation. In practice, a security model must be applied when opening a file connection and when opening a stream for the connection. MIDP 2.0 Recommended Security Policy assigns MIDlet suites to four domains based on their origin. The *Manufacturer domain* has unrestricted access to files. *Trusted 3rd party domain* and *Untrusted domain* have limited access to the following files and directories:

- The native storage directories for photos, video, and tones.
- Memory card root directory and any subdirectories of it that are not explicitly denied.
- The work directory of the MIDlet suite. (The work directory will be private for each MIDlet suite in future releases of the Nokia 6131 NFC SDK.)

### 15.2.2 Personal Information Management (PIM)

Accessing Personal Information Management (PIM) data is a common feature on many PDAs and other mobile devices. The PIM API is an optional package that gives access to PIM data residing on mobile devices. The Nokia 6131 NFC SDK includes a PIM API to provide MIDlets with access to the personal user data stored in the emulator's memory. In the Nokia 6131 NFC SDK this means the contents of the phone book, calendar entries, and items on the to-do list.

When you launch the Nokia 6131 NFC SDK's emulator, you can access personal user data by using the `javax.microedition.pim` package. The initial personal user data is stored in an emulator-specific file, for example `<SDK's installation directory>\bin\pim_db.xml`.

Modifications to the personal user data are stored in the emulator's random access memory and will be lost when you shut down the emulator.

## 15.3 Java™ APIs for Bluetooth (JSR-82)

Bluetooth is a standard for wireless integration of small devices. The API standardizes a set of Java APIs to allow devices that implement Java technology to integrate into a Bluetooth environment.

### 15.3.1 Starting emulators

Start an SDK, for example, with IDE tools, by using commands in the console window, or by any other way that is suitable for starting SDKs.

### 15.3.2 Checking the SDK phone numbers

After NCF startup, the NCF Service icon is displayed on the system tray. To check the phone numbers of the started products, click the NCF Service icon in the system tray. A pop-up menu shows the SDK names and phone numbers.

**Note:** Wait a few seconds before checking the phone numbers because the phone number initialization takes a while.

Continue to use SDKs and send messages between them by using phone numbers.

The Nokia 6131 NFC SDK offers Bluetooth emulation through NCF. A MIDlet can use Bluetooth connections if the emulator can connect to the Nokia Connectivity Framework during emulator startup.

**Note:** The Nokia 6131NFC SDK emulator does not provide a dialog for entering a PIN code during the PIN handshake. Instead, they automatically respond to any PIN code request with the character string “1234.” Use this value whenever the destination/originating host requires this information. For example, when a “Please insert PIN code” dialog is displayed.

### 15.3.3 Starting a Java application via Bluetooth

The Nokia 6131 NFC SDK includes support for “connect-anytime" services for Bluetooth through MIDP 2.0 PushRegistry. This allows you to start a Java application automatically via Bluetooth. You can use static or dynamic registration for incoming Bluetooth connections.

A service record for a URL defined in the `MIDlet-Push-<n>` attribute in the JAD file is created and installed in the Service Discovery database (SDDB) during MIDlet suite installation. A service record is also created and installed if an inbound connection is registered dynamically (with `registerConnection()`).

MIDlet suites can use `MIDlet-Push-<n>` attributes in the JAD or JAR manifest for the static registration of incoming Bluetooth connections. The emulator listens to registered Bluetooth connections and launches the required MIDlet when a connection arrives. The JAM stops listening to the push connection when the MIDlet suite that registered the connection is uninstalled or when the suite removes the connection from the PushRegistry. The following line is a sample JAD attribute:

```
MIDlet-Push-1: btl2cap://localhost:1020304050d0708093a1b121d1e1f100,
L2CAPStaticPushRegistry, *
```

The following rules apply for the static and dynamic registration of Bluetooth connections in the PushRegistry using parameters `ConnectionURL`, `MIDletClassName`, and `AllowedSender`. During static registration, these parameters are taken from the JAD file or the JAR manifest; the attribute name is `MIDlet-Push-<n>`. If dynamic registration takes place, these parameters are passed to the `registerConnection()` method of the `PushRegistry` class.

The `MIDletClassName` parameter must be used according to the MIDP 2.0 specification. The `ConnectionURL` parameter must comply with one of the JSR-82 server connection URL formats defined in the JSR-82 specification.

The following are examples of correct values for the `ConnectionURL` parameter (for L2CAP and SPP):

```
btl2cap://localhost:3B9FA89520078C303355AAA694238F08;name=Aserv
```

```
btspp://localhost:3B9FA89520078C303355AAA694238F08
```

The `AllowedSender` filter is applied to the Bluetooth address and type (authenticated or authorized) of the device that originates the connection. You can use wildcards "*" and "?" as specified in the MIDP 2.0 specification, as well as Bluetooth-specific filtering features such as blacklisting. Filtering takes place in two stages: first the connecting device passes the optional blacklist filter and then the conventional filter.

Below are some examples of a correct `AllowedSender` filter string.

The following string allows all connections except the MAC address specified in the blacklist:

```
*;blacklist=00E00379A123
```

The following string allows all authenticated connections except MAC addresses starting with "00E00379A":

```
*;authenticated;blacklist=00E00379A*
```

The following string allows all authorized connections from MAC addresses that do not match the specified blacklist:

```
*E00379A???;authorized;blacklist=??E003*;00E00379A123
```

A single `*` would allow all connections.

Registering a connection with a URL that is already registered will throw an `IOException`. When the connection is checked, the protocol and UUID are matched to the existing registered connections. URLs containing the same UUID but different protocol are considered two different connections.

The Nokia 6131 NFC SDK supports Bluetooth communication using OBEX over RFCOMM. A client connects to the Bluetooth address and channel identifier of the device, separated by a colon. A client can perform a service discovery to find out an OBEX service and its channel identifier. The server always registers the OBEX service with the service discovery database (SDPP) using a `localhost` followed by a colon and the service class UUID. The server cannot determine the channel identifier by itself. Instead, the device determines the channel to which clients can connect.

The following is a valid example string of OBEX over RFCOMM. It shows how an OBEX client connects to a MAC address and its channel identifier number 12:

```
btgoep://0050C000321B:12
```

The following is a valid example string showing how an OBEX server registers its service class UUID:

```
btgoep://localhost:12AF51A9030C4B2937407F8C9ECB238A
```

### 15.3.4 Communication with hardware devices

The Nokia 6131 NFC SDK can communicate with real hardware devices via Bluetooth. You also need a Bluetooth USB adapter in your computer. You may use, for example, Socket Bluetooth card as a Bluetooth adapter.

Nokia Connectivity Framework enables the Bluetooth communication. The Nokia Connectivity Framework 1.2 User's Guide [6] explains in a more detailed way how to install the needed Bluetooth USB drivers. After you have made the needed configurations, you can communicate between a real Bluetooth device and the Nokia 6131 NFC SDK device by using phone numbers.

## 15.4 Wireless Messaging API (JSR-120)

The Wireless Messaging API provides standard access to wireless communication resources. This JSR offers a set of reusable components that can be used individually or combined within a Java ME profile. It enables developers to build intelligent connected Java applications by using short message service (SMS) messages.

SMS messaging between emulators can be emulated by using NCF.

## 15.5 Wireless Messaging API 2.0 (JSR-205)

The Wireless Messaging API 2.0 provides standard access to wireless communication resources. This JSR is an extension to the Wireless Messaging API (JSR-120) and allows Java applications to compose and send multimedia messages (MMS), which can contain text, images, and sound. This technology allows richer possibilities for messaging on mobile devices.

## 15.6 Mobile 3D Graphics API for J2ME™ (JSR-184)

The Mobile 3D Graphics API for J2ME™ is an optional package, which is used together with several Java ME profiles, in particular MIDP. The API provides scalable, small-footprint, interactive 3D features for use on mobile devices. The Mobile 3D Graphics API can be used to create MIDlets that make use of 3D graphics, for example, in games, user interfaces, character animation and maps, and for visualization purposes.

The Nokia 6131 NFC SDK provides APIs and emulation for the Mobile 3D Graphics API.

**Note:** The Nokia 6131 NFC SDK emulator does not emulate the 3D graphics performance capabilities of an actual device. The performance of the emulator depends on the processor speed of the PC, the display adapter, and the performance of the Java virtual machine you use for running the emulator.

## 15.7 Scalable 2D Vector Graphics API for J2ME™ (JSR-226)

This optional package API is used for rendering scalable 2D vector graphics, including image files in W3C Scalable Vector Graphics (SVG) format. Most often this API is used for map visualization, scalable icons, and other graphics applications. The API is targeted for the Java ME/CLDC/MIDP platform.

## 15.8 Web Services JAXP API 1.0 for J2ME™ (JSR-172)

This JSR provides two optional packages for accessing existing Web services.

The first package is `jaxp` providing basic processing capabilities of structured XML data, and the second one is `jaxprpc` which enables XML-based RPC communication between Java ME clients and Web services.

This SDK includes only the `jaxp` optional package for XML processing.

## 15.9 Contactless Communication API (JSR-257)

The Contactless Communication API is an optional package that allows applications to access information on various contactless communication targets such as secure cards and NFC tags. NFC (Near Field Communication) is a short-range radio frequency technology that evolved from a combination of contactless radio frequency identification (RFID) and interconnection technologies. Operating over a distance of only a few centimeters, it allows users to read and write small amounts of data from tags, and to communicate with other devices, by a simple touch. When touching a tag, the NFC device reads the data stored in the tag, and initiates the appropriate action after the user's confirmation.

NFC Forum specifies a data-packaging format called NDEF to exchange information between an NFC device and another NFC device or an NFC tag. The JSR-257 API takes advantage of this packaging format by providing a connection to any physical target that supports the NDEF data formatting. The JSR-257 API allows contactless communication with external smart cards by providing a discovery mechanism for them. The actual communication with the ISO14443 compliant smart cards is done using APDU commands. For example, a real device may contain bus tickets and the external reader in the bus reads one ticket from the device by using Near Field Communication technology.

Basically the use of this API consists of two phases: discovering contactless targets and data exchange with those targets. There are different types of contactless targets. This API extends the Generic Connection Framework (GCF) architecture by defining new protocols for communicating with different kind of contactless targets. In the API, each contactless target type is located in its own API package.

To improve the usability and minimize memory consumption in the device, applications can be started after a wake-up event by using the MIDP 2.0 PushRegistry. Registration for startup is based on the record type name and format of the NDEF record. There can be one application for each record type name and format pair registered for startup at a time.

For information on how to use contactless communication targets (simulated cards) in the Nokia 6131 NFC SDK, see Chapter 3, “Using simulated cards.” Moreover, Chapter 4, “Using an external card reader,” describes the use of external card readers with the SDK.

# 16 Terms and abbreviations

| Term or abbreviation | Meaning |
|---|---|
| API | Application programming interface |
| IDE | Integrated development environment |
| JRE | Java Runtime Environment |
| JSR | Java Specification Request |
| MIDP | Mobile information device profile |
| NCF | Nokia Connectivity Framework |
| NFC | Near Field Communication |
| OTA | Over the air |
| PIM | Personal Information Management |
| SDK | Software development kit |
| SVG | Scalable Vector Graphics |
| UEI | Unified Emulator Interface |
| URI | Uniform resource identifier |

# 17 References

**Other documents:**


[1] Carbide.j 1.5 User's Guide, www.forum.nokia.com/tools

[2] Giesecke & Devrient, Sm@rtCafé® Professional Toolkit 2.0, www.gi-de.com/portal/page?_pageid=42,105616&_dad=portal&_schema=PORTAL

[3] IBM, JCOP Tools 3.0, www-306.ibm.com/software/wireless/wecos/tools.html

[4] J2ME Wireless Toolkit User's Guide (http://java.sun.com/products/sjwtoolkit/)

[5] Nokia 6131 NFC SDK: Installation Guide, http://www.forum.nokia.com

[6] Nokia Connectivity Framework 1.2 User's Guide, www.forum.nokia.com/tools

[7] NXP Product Pages, www.nxp.com/products/identification/readers/contactless/index.html

[8] OMNIKey Driver Downloads, http://omnikey.aaitg.com/index.php?id=69

[9] OMNIKey Product Pages, http://omnikey.aaitg.com

[10] Sun Java Card Development Kit 2.2.2, java.sun.com/products/javacard/dev_kit.html


**Specifications:**


JSR 135: *Mobile Media API*
Java Community Process, 2002
http://www.jcp.org/en/jsr/detail?id=135

JSR 234: *Advanced Multimedia Supplements*
Java Community Process, 2005
http://www.jcp.org/en/jsr/detail?id=234

JSR 75: *PDA Optional Packages for the J2ME™ Platform*
Java Community Process, 2004
http://www.jcp.org/en/jsr/detail?id=75

JSR 82: *Java™ APIs for Bluetooth*
Java Community Process, 2002
http://www.jcp.org/en/jsr/detail?id=82

JSR 120: *Wireless Messaging API*
Java Community Process, 2002
http://www.jcp.org/en/jsr/detail?id=120

JSR 205: *Wireless Messaging API 2.0*
Java Community Process, 2004
http://www.jcp.org/en/jsr/detail?id=205

JSR 184: *Mobile 3D Graphics API for J2ME™*
Java Community Process, 2004
http://www.jcp.org/en/jsr/detail?id=184

JSR 226: *Scalable 2D Vector Graphics API for J2ME™*
Java Community Process, 2005
http://www.jcp.org/en/jsr/detail?id=226

JSR 172: *J2ME™ Web Services Specification*
Java Community Process, 2004
http://www.jcp.org/en/jsr/detail?id=172

JSR 257: *Contactless Communication API*
Java Community Process, 2006
http://www.jcp.org/en/jsr/detail?id=257

# 18 Evaluate this resource

Please spare a moment to help us improve documentation quality and recognize the resources you find most valuable, by rating this resource.