JAVA™ MIDP APPLICATION DEVELOPER'S GUIDE FOR NOKIA DEVICES

Version 1.0

27 Nov 02

Table of Contents

1. INTRODUCTION

1.1 Purpose and Scope

Java™ MIDP Application Developer's Guide for Nokia Devices describes the Java Mobile Information Device Profile (MIDP 1.0) from the developer's point of view. It provides valuable ideas and guidelines on producing good-quality MIDP applications for Nokia devices. Readers are encouraged to check the Forum Nokia Web site at http://www.forum.nokia.com for updates to this document.

The document offers detailed information about writing MIDlets for Nokia's Java-enabled phones. Readers can learn more about Java from the Sun Microsystems Web site at http://java.sun.com/.

1.2 Audience

This developer's guide is aimed at all Forum Nokia members who are interested in producing MIDP applications for Nokia devices. It assumes familiarity with the basic issues concerning MIDP and its Application Programming Interfaces (APIs).

1.3 Additional Information

The following additional information can be found at http://www.forum.nokia.com/java:

- Documents: Java J2ME™ MIDP Documentation:
  - A Brief Introduction to MIDP Programming
  - Efficient MIDP Programming
  - Nokia API documentation
- Tools and SDKs:
  - Nokia MIDP SDKs
  - Nokia MIDP-related tools
- Code and examples:
  - Example MIDlets and tutorials
- Technical information about Nokia's Java-enabled products

A more detailed description of MIDP can be found at http://java.sun.com/products/midp.

2. JAVA MIDP

2.1 Introduction

The Mobile Information Device Profile (MIDP) is a set of Java APIs that, together with the Connected Limited Device Configuration (CLDC), provides a complete J2ME application runtime environment targeted at mobile information devices, such as cellular phones. The MID Profile provides a standard runtime environment that allows new applications and services to be dynamically deployed on end-user devices.

2.1.1 Connected Limited Device Configuration (CLDC)

CLDC defines used Java language and virtual machine features. CLDC is closely tied to a Java Virtual Machine (JVM). It defines the Java language features and the core Java libraries of the JVM.

CLDC contains the following APIs:

- Core APIs: `java.lang`
- Input/output features and handling of data streams: `java.io`
- Calendar, date, random number generation, basic data structures: `java.util`
- Basic networking features: `javax.microedition.io`

2.1.2 Mobile Information Device Profile (MIDP)

Mobile Information Device Profile (MIDP) is built on top of a CLDC. MIDP is a platform that provides a complete runtime environment for a specific kind of device. It defines APIs for user interface components, input and event handling, persistent storage, networking, and timers, taking into consideration the screen and memory limitations of mobile devices.

MIDP has the following APIs:

- User Interface: `javax.microedition.lcdui`
- HTTP[1]: `javax.microedition.io.HttpConnection`
- Persistent Storage: `javax.microedition.rms`
- Application Lifecycle: `javax.microedition.midlet`
- Timers: `java.util.Timer & java.util.TimerTask`
- One New Exception Case: `java.lang.IllegalStateException`

2.2 Developing Applications with MIDP

A MIDlet is an application written for the J2ME/MIDP platform. When designing MIDP applications for mobile phone use, developers should keep the following set of items in mind:

- Applications should be useful to users who are not necessarily experts in technology areas and devices other than the mobile phone they are currently using.
- MIDP applications should be usable in many kinds of situations, even, for example, when people are not fully concentrating on the device.

[1] In some products this is implemented using WAP protocols.

- There are different kinds of MIDP devices, so developers should consider various sizes of displays, different keyboards, and the "look and feel" of devices.
- Developers should also consider situations when a phone call or SMS comes during the execution of a MIDlet. These events cause the MIDlet to be moved to the background. When the MIDlet goes to the background automatically, it should also recover from the situation smoothly.
- In a mobile environment the network coverage varies. The device might suddenly lose network connection while a MIDlet is running. Various test cases are crucial when developing high-quality applications for mobile devices.

3. MIDP USER INTERFACE LEVELS

There are two user-interface levels in MIDP: high-level APIs and low-level APIs.

3.1 Displayable Class

A Displayable object is an element that can be placed on the display. A Displayable object may have Commands and associated CommandListeners with it. Only one Displayable at a time is active and displayed to the user. The contents displayed and their interaction with the user are defined by subclasses.

Class Displayable has two direct subclasses:

- Canvas
- Screen

Application developers may use both Canvases and Screens in the same application via special method Display.setCurrent(), which puts the required component to the display.

Figure 1 describes the major MIDP user interface classes in the class hierarchy.

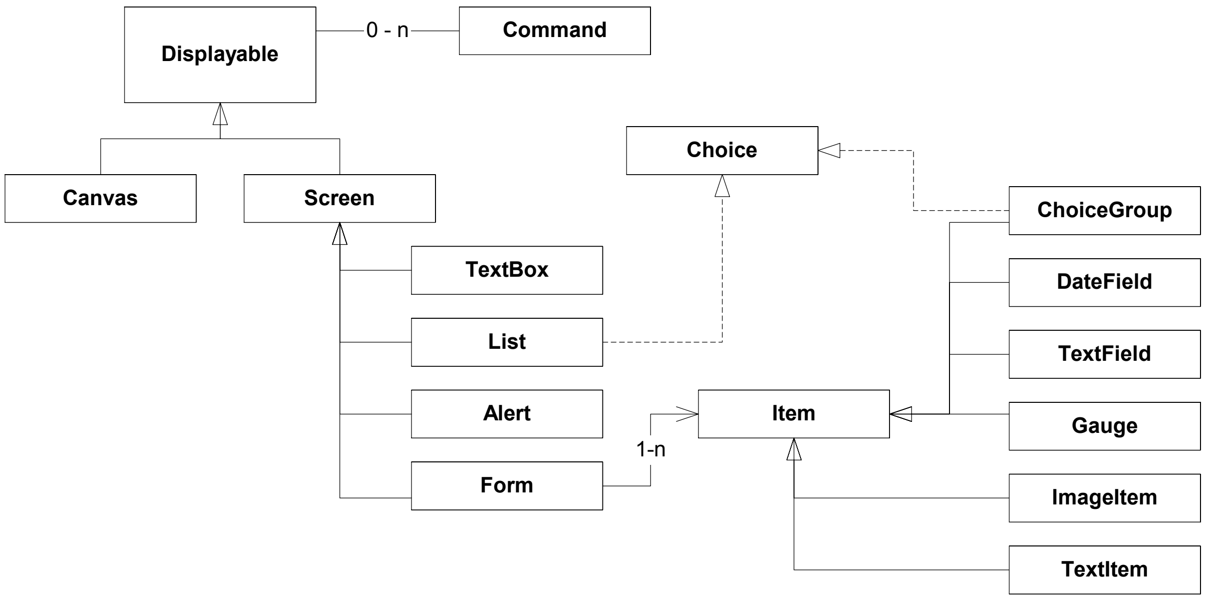


Figure 1: MIDP UI class hierarchy

3.2 High-Level APIs

High-level APIs contain a basic set of UI components for building mobile applications. Developers should use high-level API components when possible, because they make applications simpler and more portable. These components have the following properties:

- Drawing to the display is performed by the device's system software. Applications do not define the visual appearance (e.g., colors, fonts, etc.) of the components. Thus, applications are automatically using the look and feel of the device the application is running on.

- Navigation, scrolling, and other primitive interactions with the UI components are performed by the device, not by the applications.
- Applications use input mechanisms provided by the device. Low-level input is not available.

Figure 2 gives an example usage of a high-level API component.

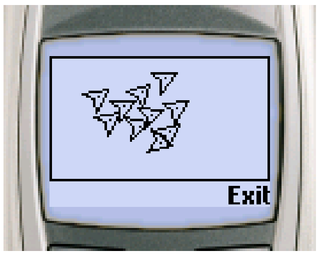


Figure 2: Example usage of high-level APIs in a MIDlet

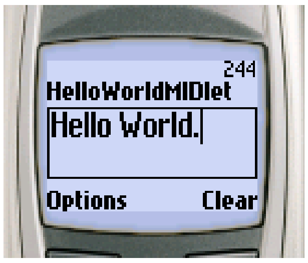


Figure 3: Example usage of low-level APIs in a MIDlet

3.3 High-Level User Interface Elements

The following sections include detailed descriptions of UI elements in MIDP—their purpose and usage. Updated user interface illustrations for specific components will be added later. Developers are encouraged to check the latest version of this document at http://www.forum.nokia.com.

3.4 Screen Class

Screen is a common super-class of all high-level user interface classes. Every Screen provides presentation and layout. MIDP applications do not define presentation or layout in Screen—just the content. Screen most probably maps directly to native UI controls with the same semantics of specific implementation.

Direct subclasses of Screen are listed below.

- Alert
- Form
- List
- Textbox
- Ticker

3.4.1 List

The List class is a Screen containing a list of choices. When a List is present on the display, the user can interact with it indefinitely (for instance, traversing from element to element and possibly scrolling). These traversing and scrolling operations do not cause application-visible events.

The three kinds of lists are as follows:

- **IMPLICIT** – where select operation causes immediate notification of the application if there is a `CommandListener` registered. The element that has the focus will be selected before any `CommandListener` for this List is called. An implicit SELECT_COMMAND is a parameter for the notification.

- **EXCLUSIVE** – where select operation changes the selected element in the list. Application is not notified.

- **MULTIPLE** – where select operation toggles the selected state of the focused element. Application is not notified.

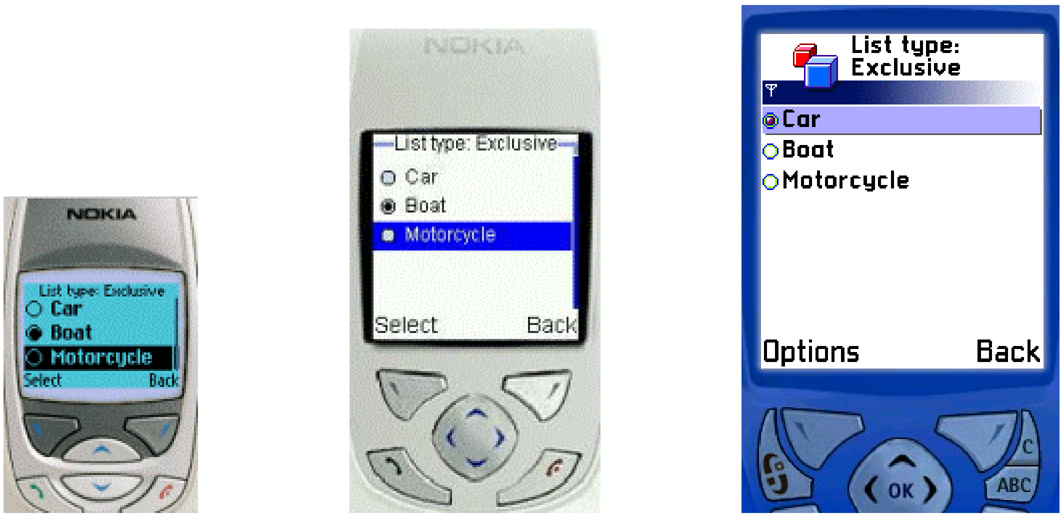


Figure 4: Exclusive List screens in Nokia 6310i, Nokia Series 40, and Nokia Series 60 emulators

3.4.2 TextBox

The TextBox class is a Screen that allows the user to enter and edit text. A TextBox has a maximum size, which is the maximum number of characters that can be stored in the object at any time (its capacity).

TextBox is a text-entry field. It can support normal text, predictive text input text (T9), or encoded text, e.g., in password cases, where letters are represented as asterisks (**).

3.4.3 Alert

Alert is a screen that shows the user various kinds of messages. An application-specific icon may be associated with it. Developers should consider the amount, tone, and text of the message carefully, and try to clarify all the possibilities in the design phase.

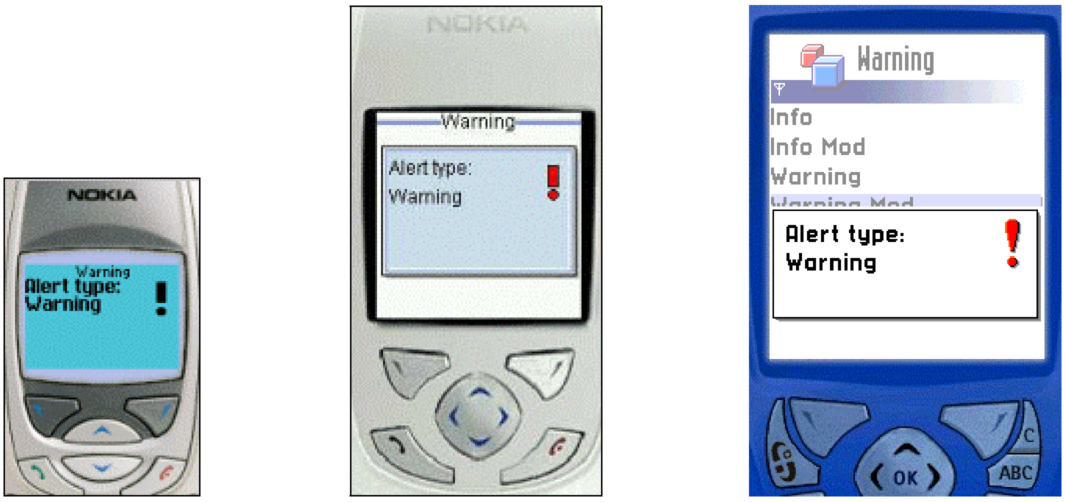


Figure 5: Alert screens in Nokia 6310i, Nokia Series 40, and Nokia Series 60 emulators

3.4.4 Form

The Form class is appropriate when a screen with a single function is not enough. Class Form can contain other user interface elements, which are called Items. Developers should keep in mind that it is not a good idea to have many different components in one display, because the size of the display is limited. Simple, clearly understandable displays with a minimum number of components are more sophisticated and usable than displays with multiple components.

The following Items can be placed inside a Form:

- **Choice group** -- used for single and multiple selections
- **Date field** – used to query times and dates
- **Gauge** – used to display a value graphically and query for a numeric value for gauge
- **Image item** – used to display images
- **String item** – used for static text
- **Textfield** – used for textual input with constraints

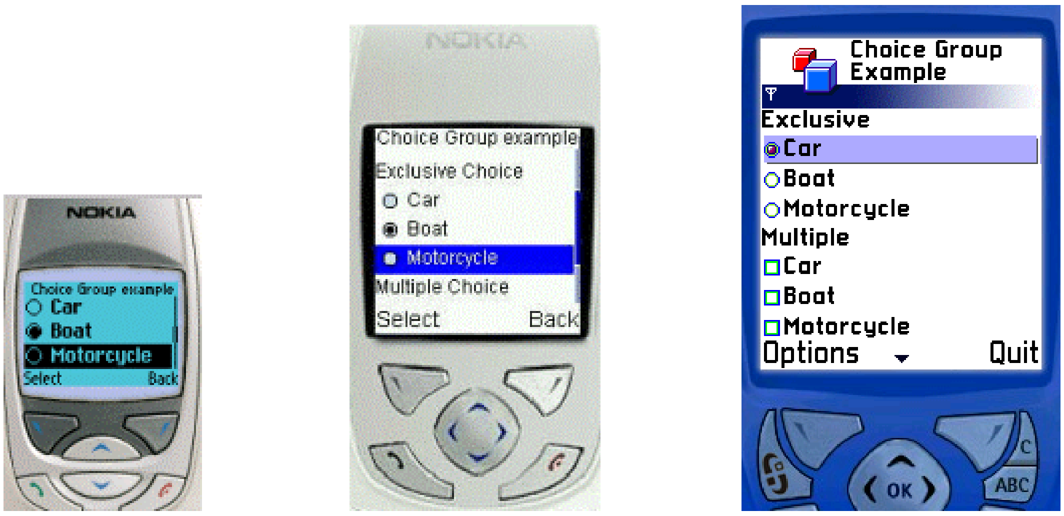


Figure 6: Exclusive Choice screens in Nokia 6310i, Nokia Series 40, and Nokia Series 60 emulators

3.5 Low-Level APIs

Low-level APIs are mainly used for games and applications that require use of interactive graphics and customizable components. The intention of the Canvas is to use customized components; for example, Canvas can be used for a clock application. It is possible to use the entire screen with the FullCanvas class of Nokia UI APIs.

The two main classes of low-level user interface are `Graphics` and `Canvas`. The `Graphics` class provides methods to paint lines, rectangles, arcs, text, and images to a `Canvas` or an `Image`. In addition to using abstract commands, as do high-level APIs, low-level APIs give developers access to key-press events. Figure 3 gives an example of the usage of low-level user interface APIs.

3.5.1 Canvas Class

The Canvas class is a base class for writing applications that need to handle low-level events and for issuing graphics calls for drawing to the display. These are similar to Graphics in J2SE™ and PersonalJava™ `(java.awt.Graphics and javax.microedition.lcdui.Graphics).`

Game applications will make heavy use of the Canvas class: it provides methods for handling game actions and key events. Canvas is a low-level component that the application defines itself; it provides a low-level graphics interface for the application to display to the screen. It is also possible to use com.nokia.mid.ui.FullCanvas for game applications. The FullCanvas class is included in the Nokia UI API.

Methods are also provided to identify the device's capabilities and keyboard mapping. In addition to games, the applications that will use Canvas are most probably other graphic-intensive applications such as location maps.

3.6 Command Class

Command is the primary mechanism for user interaction, e.g., for changing between Displayables. Command is an abstract entity for UI elements that invoke an action. Every Screen and Canvas can have an arbitrary number of Commands. Commands are mapped to the real UI elements by the MIDP device. These concrete UI elements may be soft keys, menu items, buttons, button widgets, etc.

The `Command` object has three properties:

```
commandExit = new Command(label, type, priority);
```

1. The label is shown to the user when Command is rendered on the display. It describes the function to the user.
2. The type of command can be one of the following: BACK, EXIT, STOP, HELP, OK, CANCEL, ITEM, or SCREEN.
3. Type and priority allow the implementation to make high-priority commands more accessible.

Commands have application-defined abstract types (BACK, EXIT, STOP, HELP, OK, CANCEL, ITEM, SCREEN) and priorities that are used to map commands to the correct place in a device user interface. Commands have labels that are shown to the user on the UI. If the label is empty, i.e., "", the default label is displayed, as shown in Table 1. Default labels are based on the type of the Command. After choosing the types for each Command, their importance can be set by using the priority attribute. The actual Label to the Command should be short but descriptive.

The application uses the command type to specify the intent of this command. For example, if the application specifies that the command is of type BACK, and if the device has a standard of placing the "back" operation on a certain soft key, the implementation can follow the style of the device by using the semantic information as a guide.

Command's application level handling is based on a listener function. Each Displayable object has a single CommandListener. When the user invokes a Command on a Displayable, its listener is called. To define itself as a listener, an object must implement the interface CommandListener and its method, commandAction.

MIDP Command Type (Constant Value)	**Intended Use of a Command**	**Nokia Default Placement**	**Nokia Default Command Label**
SCREEN (1)	MIDlet-defined command relevant to the current display.	LSK	Select
BACK (2)	Navigation command that should return the user to the logical previous display.	RSK	Back
CANCEL (3)	Standard negative response to a dialog on the current display.	RSK	Cancel
OK (4)	Standard positive response to a dialog on the current display.	LSK	OK
HELP (5)	Request for MIDlet help information about the current application or the current Displayable.	LSK	Help
STOP (6)	Stops a currently running process or operation.	RSK	Stop
EXIT (7)	Exits the MIDlet. Application EXIT Command should call the MIDlet's notifyDestroyed() method, after doing needed resource cleanup and state-saving operations.	RSK	Close/Exit
ITEM (8)	Command that should be used for tasks that relate to the focused/selected content of the Displayable.	LSK	Select

Table 1: MIDlet Commands and their intended functions; default placements in Nokia Series 30 and Series 40 phones (LSK = Left Soft Key, RSK = Right Soft Key); and the Nokia default Command labels

3.6.1 Command Mapping

Generally in Nokia mobile phones, Commands are mapped to the soft keys and to the Options menu. In Nokia Series 80 devices, Commands are mapped to the command bar buttons and menus.

Table 1 summarizes information for the Command type fields. The "Intended Use of a Command" column offers the normal interpretation of the command given; however, there is no automatic implementation, and the action to be taken is defined by the application's public void commandAction (Command c, Displayable d) method. The type of the Command is used for deciding the placement of the Command in the user interface of a particular device. The table also indicates which soft key the Command would normally be mapped to. Default soft-key texts are provided, to be used if the MIDlet assigns a label that is an empty string or a string containing just white space. An exception is thrown if the MIDlet assigns a null label.

3.6.2 Details of Command Mapping in Nokia Devices

All Commands within a Displayable are placed in a device user interface as follows.

Right Soft Key (RSK)

First, RSK is filled with a single Command based on types of the Commands available in Displayable. One Command from the following Command types will be placed to RSK in this priority order:

1. STOP
2. CANCEL
3. BACK
4. EXIT

If there is more than one Command with the *same selected type* (e.g., there are two CANCEL Commands but no STOP Command), then Command priority is used to select the one that appears in RSK. This also means that if there are two Commands, for example, with STOP and CANCEL types, and CANCEL has higher priority value in the Command object, then STOP will be displayed on RSK as it has higher priority *type*.

Left Soft Key (LSK)

The remainder of the Commands will be placed under the Options menu available from LSK. This means that LSK has the label "Options," which opens a selection list of Commands.

If there is only a single OK, ITEM, or SCREEN Command available in the Options menu, the Command is placed directly to LSK and the Options menu is not used. Note also that in some Screens there are implementation-provided operations available in the Options menu that may prevent the direct use of LSK. For example, in TextBox there may be predictive text input related operations in the Options menu.

Ordering of Commands in Options menu

The order of Commands in the Options menu is first based on type and then based on priority:

1. STOP Commands
2. OK Commands
3. CANCEL Commands
4. ITEM Commands
5. SCREEN Commands
6. HELP Commands
7. BACK Commands
8. EXIT Commands

Note also that the label "Options" is localized based on currently selected user interface language.

3.6.3 Important Design Notes about Commands

- Avoid large numbers of Commands because for example in Nokia Series 30 there is a single menu where commands are shown. Having a large number of Commands in the Options menu makes the user interface inconvenient to use.

- In the design phase, break the application into views and design interaction between each view. Screens and Commands can then be used to realize the actual tasks in views.
- The most commonly used functions should be the first items on the list, and so on. Always use correct Command type as this place the Commands in the correct order in menus and elsewhere in the user interface. It is always suitable to provide an **EXIT** command to the user so that he/she can exit the application smoothly.
- Limit the length of Command labels; make the descriptions as clear and short as possible. Put the most important things in the beginning. Developers should remember that labels can be shortened differently in different devices. Consider, for example, the label "Write greeting." The user sees the following text on the screen: "Write greeting" => "Write..." in the real UI. It is important to note that in Nokia devices there are certain behavioral rules that are already familiar to users. For example, if there are two soft keys in the device (left and right keys), the right soft key provides a "Way back" or "Exit" function to the user. Developers should respect these conventions when designing easy-to-use applications for Nokia devices. In MIDP implementation this is handled by Command mapping: "negative" backwards actions are always mapped into the right soft key and "positive" actions to the left soft key.

3.6.4 Events and Game Actions in Low-Level UI

`Commands` make user interaction possible in high-level user interface APIs, while their counterparts in low-level user interface APIs are called events and actions.

Key events in Canvas

When an application needs to handle key events in Canvas, it must override the Canvas methods keyPressed, keyReleased, and keyRepeated. After a key is pressed, the keyPressed method is called with the key code. When that key is released, the keyReleased method is called. Also in all MIDP devices, there must be support for the ITU-T telephone keys. Key codes are defined in the Canvas class for the digits 0 through 9, *, and #.

Game action keys in Canvas

The `Canvas` class has methods for handling game action events (also known as game actions) from the low-level user interface. The API defines actions as follows: UP, DOWN, LEFT, RIGHT, FIRE, GAME_A, GAME_B, GAME_C, and GAME_D. The device maps the action's events to suitable key codes on the device with the `Canvas.getGameAction` - method.

4. MIDLET UI DESIGN

4.1 Nokia User Interface Styles

This document examines MIDP issues from four different Nokia user interface categories. The following products are examples of current Nokia Java-enabled devices:

- Nokia Series 30 (e.g., Nokia 6310i)
- Nokia Series 40 (e.g., Nokia 7210)
- Nokia Series 60 (e.g., Nokia 7650)
- Nokia Series 80 (e.g., Nokia 9200 Communicator series)

After reading this document, developers should be familiar with these styles and how to produce MIDP applications that fit into these Nokia user interface styles.

More information about Nokia's Java-enabled products can be found at http://www.forum.nokia.com/: Device Specifications.

4.1.1 Nokia Series 30 User Interface

Nokia Series 30 is a UI concept with a monochrome or color display resolution of 96 x 65 pixels. The Nokia Series 30 UI is based on a two-soft-key concept — that is, that there are left/right soft keys, a **Send** key, an **End** key, and scrolling keys. The principle in Nokia Series 30 navigation is that the left side (left soft key, **Send** key) is used for positive actions such as accepting and selecting. The right side (right soft key, **End** key) is used for negative actions such as clearing and moving backwards.

Standard hardware keys in Nokia Series 30:

- Down and Up scroll keys
- Left and Right soft keys labelled with text on the display
- **Send** and **End** keys
- ITU-T numeric keys 0 - 9, *, #
- Power key
- Volume up/down keys

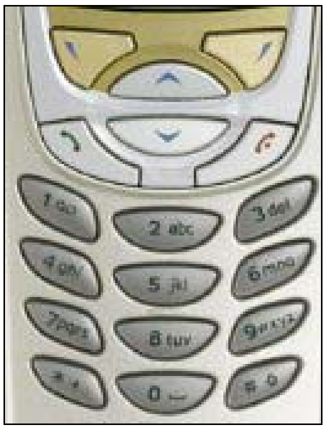


Figure 7: Standard keys in Nokia Series 30

Scroll keys in Nokia Series 30:
The scroll keys allow effective navigation in the menu tree.

Soft keys in Nokia Series 30:
The left soft key is basically used as a yes/positive key. It contains options that execute commands and go deeper into the menu structure: **Select**, **OK**, **Options,** etc. The right soft key functions as a no/negative key. It contains options that cancel commands, delete text, and go backwards in the menu structure: **Back**, **Exit**, **Clear**. The user should be able to use the phone as much as possible with just the scroll keys and the left soft key. When a soft key provides a function, it must be labelled. The same function cannot be available in both soft keys at the same time.

Send and **End** keys (call handling keys) in Nokia Series 30:
The **Send** key is mainly used for answering an incoming call and initiating an outgoing call, but can also be used as a "Select button" in various cases. The **End** key is mainly used for terminating an ongoing call, rejecting an incoming call (GSM), and as global exit key, which closes the active application and brings the application list visible.

Display components:
Nokia Series 30 supports a 96 horizontal x 65 vertical pixel display. The display is split into the following main areas:

- The Status window is for displaying status indicators, e.g., remaining battery level. The Status window includes the header text.
- The Main window is for displaying menu lists and application-specific data.
- The Soft-key window is for displaying soft-key functions.

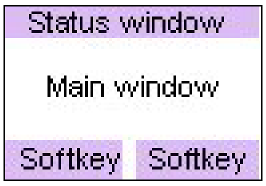


Figure 8: Nokia Series 30 UI display split into Status window, Main window, and Soft-key window

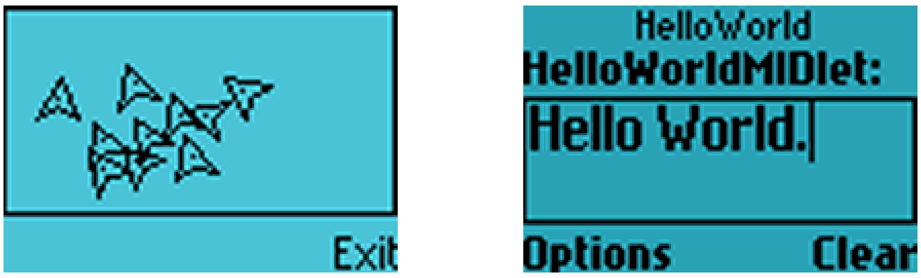


Figure 9: Nokia Series 30 UI example, with 96 x 95 pixel display

Button mappings, basic navigation:
Navigating the MIDP high-level UI elements is done with Up and Down keys located below the phone display.

The **Send** key (green phone receiver key) can also be used as Select. Commands get mapped to the two soft keys below the screen. Soft-key labels are visible only when using high-level UI elements.

Game action mapping:

Key 2 and Up arrow key	-> UP
Key 8 and Down arrow key	-> DOWN
Key 4	-> LEFT
Key 6	-> RIGHT
Key 5 and **Send** key	-> FIRE
Key 7	-> GAME_A
Key 9	-> GAME_B
Key *	-> GAME_C
Key #	-> GAME_D

Key constants not defined in MIDP specification:

Up arrow key:	-1
Down arrow key:	-2
Send key:	-10

4.1.2 Nokia Series 40 User Interface

Nokia Series 40 is a size- and cost-driven UI platform for the mass market. It has a high-resolution color display and supports four-way scrolling. Screen size is 128 x 128 pixels and 4096 colors are supported. The Nokia Series 40 UI represents the smooth evolution of the Nokia two-soft-key UI, with the proven usability of previous Nokia phones. The color display is used to improve key applications like MMS, picture viewing, Java MIDP, Browser, Calendar, and personalization, and also to increase usability of all applications.

Standard hardware keys in Nokia Series 40:

- Left soft key (LSK, key event LEFT_SOFTKEY (-6))
- Right soft key (RSK, key event RIGHT_SOFTKEY (-7))
- Up and Down scroll keys (key events KEY_SCROLL_UP (-1) and KEY_SCROLL_DOWN (-2))
- Left and Right scroll keys (key events KEY_SCROLL_LEFT (-3) and KEY_SCROLL_RIGHT (-4))
- **Send** key (key event SEND (-10))
- **End** key (key event END (-11))
- Numeric keys (0 - 9, *, #)
- Power key
- Volume Up/Down keys

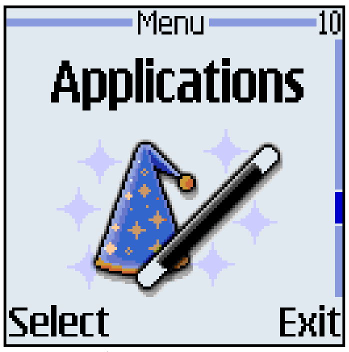


Figure 10: Nokia Series 40 UI examples, with 128 x 128 pixel display and color support

In Nokia Series 40 devices there are two places were applications can be saved — within Applications or in Games menus. The place where an application is stored depends on a string within the JAD file. Nokia has introduced a JAD attribute called Nokia-MIDlet-Category. If this is used by a developer and set to Game, then the application will be saved within the Games menu. All other applications that do not contain this attribute will be saved within the Application menu.

4.1.3 Nokia Series 60 User Interface

Nokia Series 60 is a UI style for imaging phones. It is a one-hand-operated Symbian OS feature platform. In Nokia Series 60 there is a large color display that is well-suited to different kinds of applications; display size is 176 x 206 pixels. The first product to use Nokia Series 60 UI style is Nokia 7650.

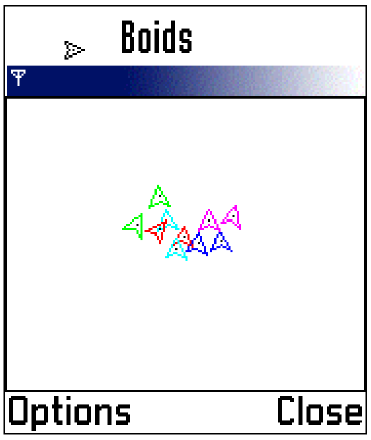


Figure 11: Nokia Series 60 UI examples, with 176 x 206 pixel display and color support

Standard hardware keys in Nokia Series 60:
- Two soft keys
- 5-function Select key (Up, Down, Left, Right,and Select)
- **Send** and **End** application keys
- **Clear**
- Shift (ABC)
- Numeric keypad (0-9, *, #) Voice key (dialing, recording) Power key

Select key concept in Nokia Series 60:
- Movements in four directions and selection
- The principal navigating method
- Twisting up/down:- Move focus

- Browse lists
- In editor: move cursor

- Clicking:

- Open current item to next lower level
- When can't open, offer Roller Options menu

Figure 12: Principle of Select key. Component and its implementation in Nokia 7650

Command mapping:

The implementation will add an Exit command into the Options menu automatically. For this reason it is sometimes possible to see two Exit commands in the Options menu.

Form:

- Commands "Copy text" and "Find in page" are automatically added into the Displayable.

Button mappings, basic navigation:

Navigating the MIDP LCDUI high-level UI elements is done with the navigation keys (Up, Down, Left, Right). OK can be used as Select. Commands are mapped to the two soft buttons below the screen.

Game action mapping:

• Key 2 and Up key	-> UP
• Key 8 and Down key	-> DOWN
• Key 4 and Left key	-> LEFT
• Key 6 and Right key	-> RIGHT
• Key 5 and OK key	-> FIRE
• Key 7	-> GAME_A
• Key 9	-> GAME_B
• Key *	-> GAME_C
• Key #	-> GAME_D

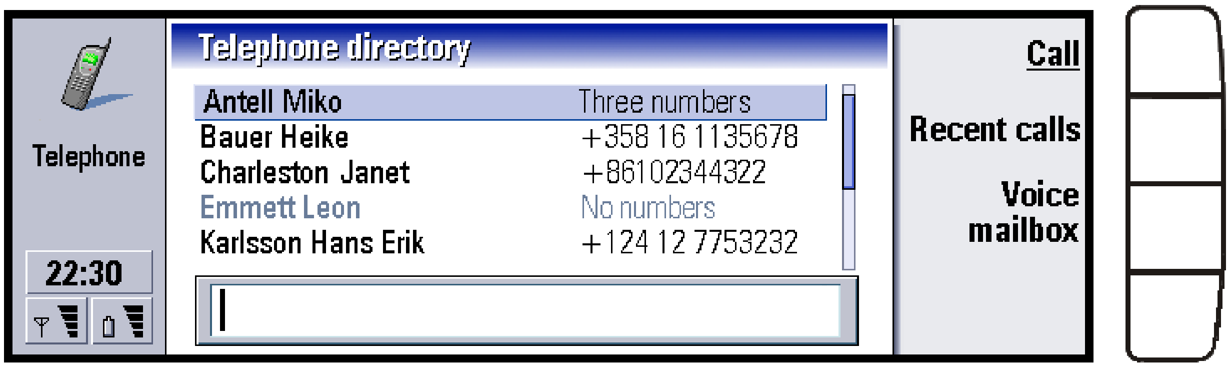
4.1.4 Nokia Series 80 User Interface

MIDP software for Nokia 9200 Communicator series is available at the Forum Nokia Web site. This software is a beta version and is usable in the device only, not in the emulator.

Nokia Series 80 is a two-hand-operated "feature concept" platform. There is a color screen and abundant space for different types of applications. The style guide for Nokia 9200 Communicator series can be found in *Nokia 9200 Communicator Series SDK for Symbian OS*.

The main interaction elements in Nokia Series 80:

- Keyboard residing below the display
- Four command buttons on the right side of the display
  - The Command Button Area (CBA) consists of four command buttons
  - The basic functionality of the device should be managed by the command buttons
  - The width of the command-button area is dynamic but view dependent; it is as wide as the longest command text string that could appear on the application screen in question
  - The first word of a command text starts with a capital letter. Use of abbreviations is not recommended. Command texts are aligned to the right. Clear, brief commands should be used, and commands should be prioritized so that only the most important functions are available via command buttons.
- Menu
  - The menu button activates and deactivates the menu



Antell Miko	Three numbers
Bauer Heike	+358 16 1135678
Charleston Janet	+86102344322
Emmett Leon	No numbers
Karlsson Hans Erik	+124 12 7753232

Figure 13: The Nokia Series 80 UI application area is in the middle of the screen; it consists of a title, margins, and main area (in the example above there is a list component in the main area).

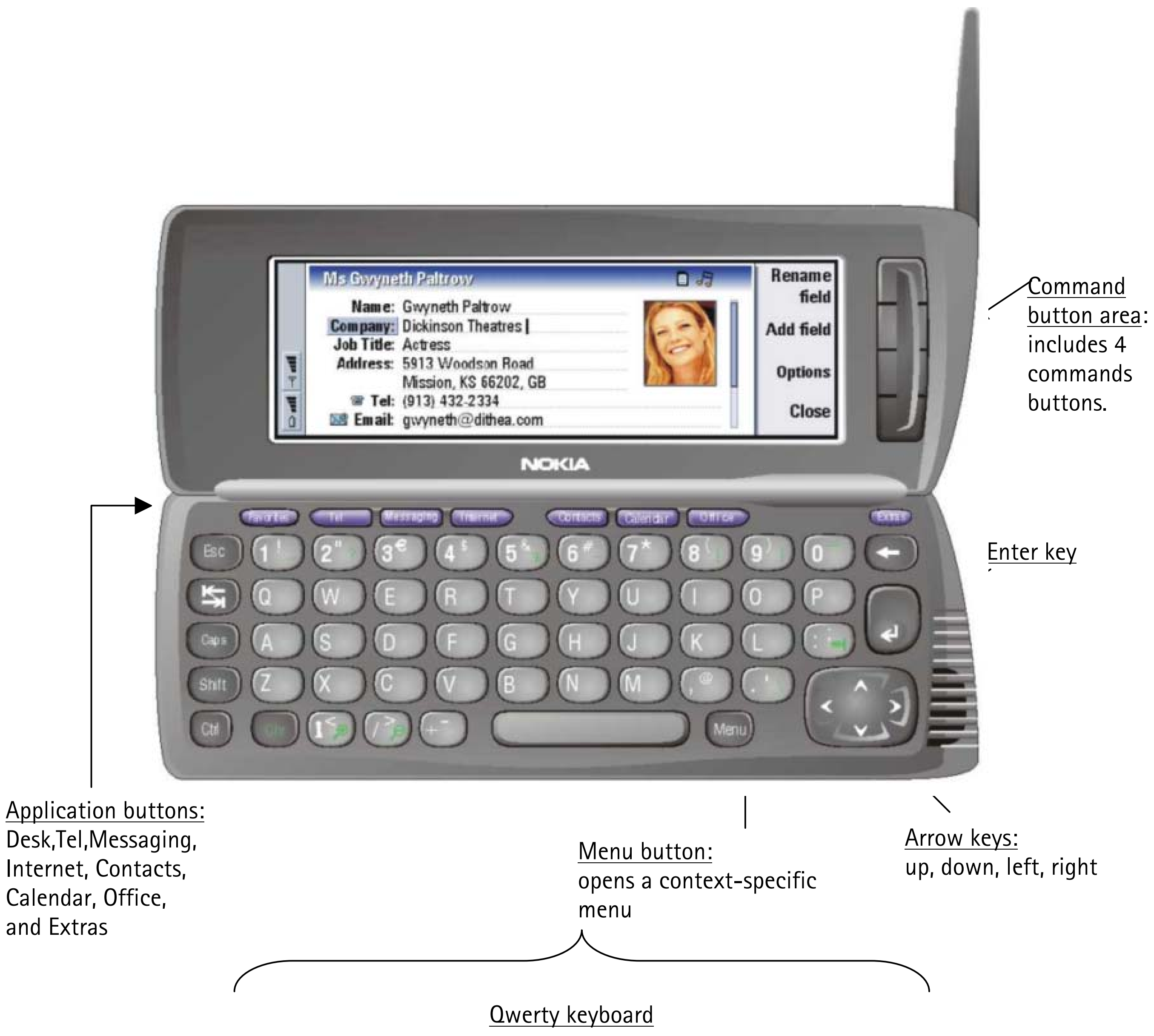

Command button area: includes 4 commands buttons.
Enter key
Application buttons: Desk,Tel,Messaging, Internet, Contacts, Calendar, Office, and Extras
Menu button: opens a context-specific menu
Arrow keys: up, down, left, right
Qwerty keyboard
Ms Gwyneth Paltrow
Name: Gwyneth Paltrow
Company: Dickinson Theatres
Job Title: Actress
Address: 5913 Woodson Road Mission, KS 66202, GB
Tel: (913) 432-2334
Email: gwyneth@dithea.com
Rename field
Add field
Options
Close

Figure 14: Nokia Series 80 keyboard

4.2 Fonts in Different Products

According to the MIDP specification, the Font class represents fonts and font metrics. Fonts cannot be created by applications. Instead, applications query for fonts based on font attributes and the system will attempt to provide a font that matches the requested attributes as closely as possible.

MIDP Font attributes to Nokia font mapping apply to MIDlet Font objects used in Graphics objects. Graphics objects' drawChar, drawChars, drawString and drawSubstring methods draw text based on the current active Font in the Graphics context. Table 2 below lists the default Fonts in Nokia devices.

	Series/ Models	Font Face	Font Style	Font Size SMALL	Font Size MEDIUM (default)	Font Size LARGE
Series 30	3410	SYSTEM (0)	PLAIN (0)	11	13	16
	3510i	SYSTEM (0)	PLAIN (0)	9	11	13
	3585	SYSTEM (0)	PLAIN (0)	11	11	13
	3590	SYSTEM (0)	PLAIN (0)	11	11	13
	6310i	SYSTEM (0)	PLAIN (0)	8	11	13
Series 40	6610	SYSTEM (0)	PLAIN (0)	9	16	23
	7210	SYSTEM (0)	PLAIN (0)	9	16	23
Series 60	3650	SYSTEM (0)	PLAIN (0)	14	15	16
	7650	SYSTEM (0)	PLAIN (0)	14	15	16
Series 80	9210	SYSTEM (0)	PLAIN (0)	12	19	
	9210i	SYSTEM (0)	PLAIN (0)	12	19	
	9290	SYSTEM (0)	PLAIN (0)	12	19	

Table 2: Default fonts in Nokia Java-enabled phones

Any style combinations can be used in Fonts. However, they are displayed differently in different UI series and devices. Also, there are some differences in font sizes depending on the languages used.

4.3 MIDP Applications in Different Products

The main goal and promise of Java is portability: the "Write once; run everywhere" principle. In practice there are many challenging aspects to portability because devices are different. Accordingly, this document is written from three different UI points of view: Nokia Series 30, Nokia Series 40, and Nokia Series 60. Here are examples of Nokia user interfaces in different products:

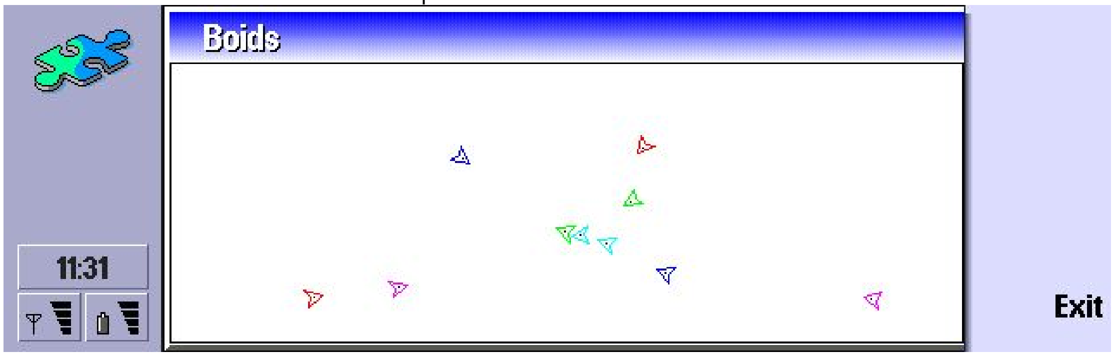


Figure 15: Boids MIDlet in Nokia 9210 Communicator (Series 80)

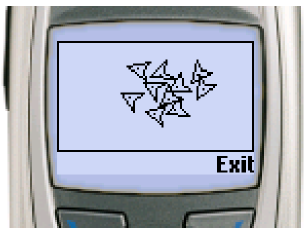


Figure 16: Boids MIDlet in Nokia 6310i (Series 30)

4.4 Designing User Interfaces

When designing user interfaces for mobile phones, developers should consider the following:

1) *Target group.* Designers should have a clear vision of their application target users (and target devices as well) before actual design and implementation begins.
2) *Early focus on users.* Designers should have direct contact with intended or actual users to make sure that they are fulfilling user requirements. In practical terms, applications should be simple, easy, and comfortable to use.
3) *Early and continual user testing.* Designers should conduct sufficient testing so that they can detect usability errors as early as possible.
4) *Integrated design.* All aspects of usability (e.g., user interface, help system, documentation) should evolve in parallel, rather that sequentially.

After developers know the end users well enough, it is time for them to produce the actual application. Here are some practical design tips:

1) Use a "screen map" or some other modeling technique to visualize your design. Split the application into the various states and clarify the connections between states (starting point and end point). In this phase, consider the issue of navigation throughout the entire application, and make sure that users can always go back.

2) Proceed to a more detailed level. Think about the different messages that you want to give to the user in various situations. Think also about all possible errors!
3) Remember that your MIDlet can go to a variety of Nokia devices, and that the screens, shapes, and sizes of devices may vary. Test as much as possible with different Nokia MIDP devices.
4) Take advantage of the MIDP high-level user interface when possible, so that your MIDlet behaves like the phone's own applications.
5) Remember to define and implement comprehensive Instructions or Help into the application, so that users can immediately find answers to their questions without calling a help desk.

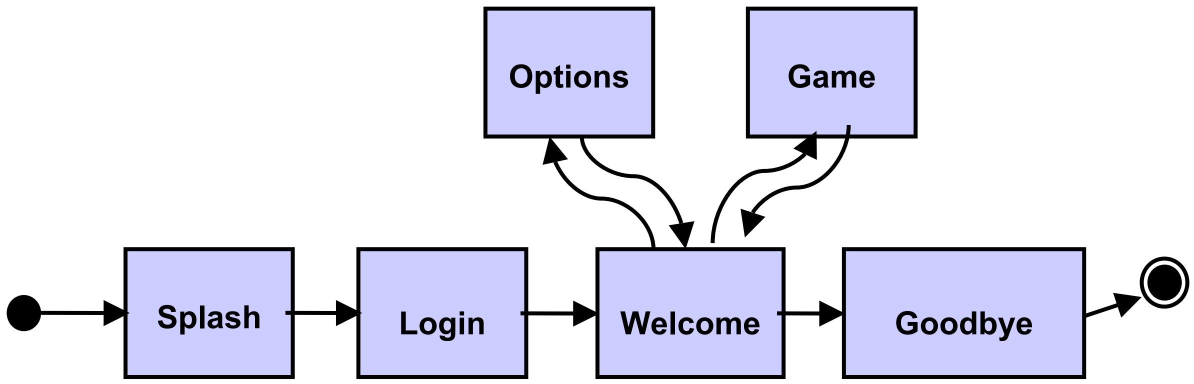


Figure 17: Screen map technique example

In addition to reading this document, developers are encouraged to access documentation and examples from the Forum Nokia Web site at http://www.forum.nokia.com: Technologies: Java. Developers should also consider carefully different methods of navigation and input mechanisms of different devices.

4.5 MIDP Application Localization

Developers should consider the localization of their MIDP applications. Localization means adapting the application to other languages and cultures — generally, it means translating user interface texts and possibly changing images or colors.

To help in the creation of different language versions of an application, the user interface texts should be isolated from the program source code and put into separate text files, also known as resource files. In MIDP, the format of these files and the mechanism used to read the correct UI texts at runtime are the responsibility of the application developer. The MIDP specification does not address these issues.

One solution is to place the resource files inside the JAR file, in the /res folder. They can be read at runtime using the getResourceAsStream() method of the current class. MIDP applications can query the system's locale, so that developers can read the correct UI texts at runtime if the texts are partitioned into one file per locale and named appropriately. The microedition.locale system property contains a string that indicates the current locale. This property is retrieved using the System.getProperty() method.

Table 3 collects the possible values of the microedition.locale system property in Nokia's non-Symbian products, such as Nokia 6310i and Nokia 3410.

	System Property	**Return Value**
CLDC	microedition.configuration	Returns configuration string, e.g., CLDC 1.0
	microedition.encoding	The character encoding used in I/O APIs if the MIDlet doesn't explicitly indicate character encoding, e.g., UTF-8
	microedition.platform	The name of the platform or device and software version, e.g., Nokia6310i/4.70
	microedition.profiles	The J2ME profiles that the device supports, e.g., MIDP 1.0
MIDP	microedition.locale	Return value MUST contain the language and can optionally contain the country code, e.g., en-US
	microedition.profiles	Returns the MIDP profile, e.g., MIDP 1.0

Table 3: System properties that can be asked using System.getProperty() method

These properties can be employed using the following method, e.g., language information: `System.getProperty("microedition.locale").`

UI Language	**microedition.locale**
American English	en-US
American Spanish	es-US
Arabic	ar
Brazilian Portuguese	pt-BR
Bulgarian	bg-BG
Canadian French	fr-CA
Chinese Hong Kong	zh-HK
Chinese Simplified	zh-CN
Chinese Traditional	zh-TW
Croatian	hr-HR
Czech	cs-CZ

Danish	da-DK
Dutch	nl-NL
English	en
Estonian	et-EE
Finnish	fi-FI
French	fr
German	de
Greek	el-GR
Hebrew	he-IL
Hindi	hi-IN
Hungarian	hu-HU
Icelandic	is-IS
Indonesian	id-ID
Italian	it
Latvian	lv-LV
Lithuanian	lt-LT
Malaysian	ms-MY
Norwegian	no-NO
Polish	pl-PL
Portuguese	pt-PT
Romanian	ro-RO
Russian	ru-RU
Serbian	sr-YU
Slovak	sk-SK
Slovene	sl-SL
Spanish	es-ES
Swedish	sv
Tagalog	tl-PH
Thai	th-TH
Turkish	tr-TR
Ukrainian	uk-UA
Vietnamese	vi-VN

Table 4: The return strings for the System.getProperty("microedition.locale") in non-Symbian products

UI Language	microedition.locale
Bahasa Indonesia	id-ID
Bahasa Malaysia	ms-MY
Bulgarian	bg-BG
Chinese, PRC	zh-CN
Chinese, HK	zh-HK
Chinese, SG	N/A
Chinese, TW	zh-TW
Croatian	hr-HR
Czech	cs-CZ
Danish	da-DK
Dutch	nl-NL
English	en
Estonian	et-EE
Finnish	fi-FI
French	fr
German	de
Greek	el-GR
Hungarian	hu-HU
Icelandic	is-IS
Italian	it
Latvian	lv-LV
Lithuanian	lt-LT
Norwegian	no-NO
Polish	pl-PL
Portuguese	pt-PT
Romanian	ro-RO
Russian	ru-RU
Serbian (Latin)	sr-YU
Slovak	sk-SK
Slovene	sl-SI
Spanish	es-ES
Swedish	sv
Tagalog	tl-PH
Turkish	tr-TR
Ukrainian	uk-UA

Table 5: The return strings for the System.getProperty("microedition.locale") in Symbian products

5. MIDP USABILITY – GUIDELINES FOR DESIGN

5.1 General Usability

Usability means that a person using the application can accomplish tasks quickly and easily.

1) Usability means focusing on users.
2) People use products to be productive.
3) Users are busy people trying to accomplish tasks.
4) Users decide when the product is easy to use.

5.2 Applications for Mobile Use

When creating applications for mobile use, designers should explain:

- Nokia style of designing MIDlets.
- Usability aspect. Think about the end user while design and implementation are ongoing (very early phase).
- Usability checklist, tips, and human-centered design methods. Define the application target user, how to perform usability tests easily, etc.

The first step in writing MIDlets is to identify the application end users. What do they want and how do they want to achieve it? The following questions can be used to determine the general usability of an application:

1) Is it easy to learn?
2) Is it efficient to use?
3) Is it easy to remember?
4) Can the user recover quickly from errors?

5.3 MIDP Application Design Checklist

There are certain parameters to keep in mind in developing user interfaces:

- Variety of display sizes and shapes
- Color or monochrome displays
- Different input mechanisms in different devices and different "look and feel" of devices
- Image behavior in displays of different sizes
- Testing of code
- Backward navigation
- Use of titles and labels
- Usability test

6. NOKIA UI API

6.1 Nokia UI API in Brief

MIDP 1.0 was designed for maximum portability and concentrates only on features that can be implemented by all candidate devices, which varying widely in capabilities. As a consequence, MIDP 1.0 excludes sound features and advanced graphics features, which makes it more portable for a wide range of devices.

A major use of Java in mobile devices is in the area of entertainment, for example, games. Games often need low-level access to device hardware such as keys and a display. MIDP was designed to be hardware-independent, so there is no support for this kind of low-level access.

Nokia UI API extensions enhance the developer's ability to deliver good Java applications for Nokia devices supporting MIDP. Currently, the Nokia UI API has two packages containing interfaces and classes for providing simple sounds and more advanced graphics:

- `com.nokia.mid.ui`:
  Contains some graphics-related extensions for MIDP low-level UI APIs, such as Graphics and Canvas.
  - public interface DirectGraphics
    DirectGraphics contains some graphics extensions for MIDP Graphics, with which polygons and triangles can be drawn and filled, images can be rotated or flipped, alpha channel color can be supported, and raw pixel data can be directly obtained from the graphics context or drawn to it.
- `com.nokia.mid.sound`:
  Provides an API for simple audio capabilities.
  - public interface SoundListener
    This interface is used by applications that need to receive events that indicate changes in the playback state of the Sound objects.

6.2 User Interface Extensions

The Nokia UI API adds to MIDP the following graphics-related features:

- Full-screen Canvas for extending the normal painting area
- Ability to draw and fill triangles and polygons
- Ability to draw images reflected or rotated
- Transparency support
- Extra ways to create mutable images
- Low-level access to image pixel data

6.2.1 Full-Screen Canvas

The screen shots in Figure 18 show the difference in screen sizes. On the right is a screen that uses the Nokia UI APIs Full Canvas class, which extends the original Canvas in order to use the entire screen for graphics. Commands cannot be added to the FullCanvas because there is no space for soft-key labels. An attempt to add commands to the FullCanvas will throw an exception.

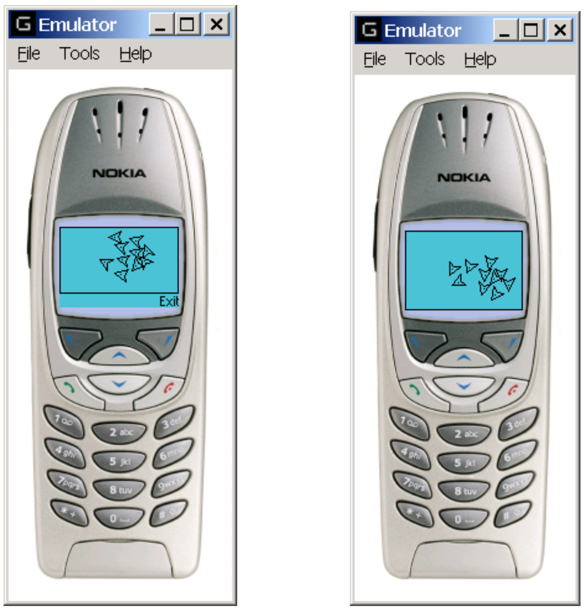


Figure 18: Difference in screen appearances when using FullCanvas

6.2.2 Drawing and Filling Triangles and Polygons

The API provides methods for drawing and filling triangles and polygons. The following example is for filling a polygon:

```
public void fillPolygon(int[] xPoints, int xOffset,
  int[] yPoints, int yOffset,
  int nPoints, int argbColor)
```

This method fills a closed polygon defined by the arrays of the x and y coordinates. It draws the polygon defined by nPoint line segments, where the first nPoint - 1 line segments are line segments from (xPoints[xOffset + i - 1], yPoints[yOffset + i - 1]) to (xPoints[xOffset + i], yPoints[yOffset + i]), for 1 <= i <= nPoints. The figure is automatically closed by drawing a line connecting the final point to the first point if those points are different.

6.2.3 Drawing Images Reflected or Rotated

Flipping and rotating existing images can be accomplished using manipulation parameter in the method drawImage:

```
public void drawImage(javax.microedition.lcdui.Image img,
int x, int y,
int anchor, int manipulation)
```

This draws an image to the graphics context and does common image manipulations during the drawing of an image. Manipulation can be 0 if no manipulation is done. Manipulation means flipping or rotating values, like DirectGraphics.ROTATE_90 (constant for rotating an image 90 degrees counterclockwise), or a combination.

6.2.4 Additional Ways to Create Mutable Images

The following methods create mutable images:

```
public static Image createImage(byte[] imageData,int imageOffset,
int imageLength)
```

This creates a mutable image that is decoded from the data stored in the specified byte array at the specified offset and length. The data must be in a self-identifying image file format supported by the implementation, e.g., PNG. Note that the semantics of this method are exactly the same as Image.createImage(byte[],int,int), except that the returned image is mutable.

Parameters:
imageData - the array of image data in a supported image format
imageOffset - the offset of the start of the data in the array
imageLength - the length of the data in the array

6.2.5 Low-Level Access to Image Pixel Data

This provides the possibility of getting and setting pixels. The example below illustrates the usage of methods for getting pixels and drawing pixels.

```
public void getPixels(int[] pixels, int offset, int scanlength,
                      int x, int y, int width, int height, int format)
```

It copies the pixel values of the graphics context from a specific location to an array of int values. The pixels will be passed in the format defined by a format parameter. If an implementation does not support the format, an IllegalArgumentException is thrown. Note that it is possible that only the native format is supported via the appropriate version of the getPixels method. This method returns only int-based formats. Requesting all other formats will result in an IllegalArgumentException.
Throws ArrayIndexOutOfBoundsException if the array size is too small for image pixels.

The getPixels method stores the pixel data to the pixels array in the following fashion:

```
pixels[offset + (x1 - x) + (y1 - y) * scanlength] = P(x1, y1),for each P(x1, y1),
where (x <= x1 < x + width) and (y <= y1 < y + height).

public void drawPixels(int[] pixels, boolean transparency, int offset,
int scanlength, int x,
int y, int width, int height, int manipulation, int format)
```

This copies or draws the pixel data directly to the graphics context to a specific location from the array starting from the specified offset. The pixels are passed in the format defined by the format parameter. If an implementation does not support the format, an IllegalArgumentException is thrown.

6.3 Sound Extensions

The Nokia UI API adds to MIDP the ability to play sound. There are two types of sound supported:

- Single notes, specified by frequency and duration
- Simple tunes, specified using the Nokia Smart Messaging ringing tone library format, which is the same that is used for downloading new ringing tones to Nokia phones

To play a single note:

```
Sound s = new Sound(440, 1000);  //frequency and time in ms.
s.play(1);                       //loop, plays the note once.
```

To play a simple tune:

```
byte[] tune =
    {
        (byte)0x02, (byte)0x4A, (byte)0x3A, (byte)0x80,..
    };
Sound s = new Sound(tune, Sound.FORMAT.TONE)
s.play(2)             //plays the tune twice
```

SoundListener Interface:

SoundListener can be used for handling sound state notifications. It is very helpful when checking the state, e.g., whether or not the previous tone stopped, before playing the new one. The following code example shows the basic model for playing sounds using the Nokia UI API:

```
    private void playSound(int priority, byte[] bytes)
    {
        if (!isPaused && (priority > currentSoundPriority))
        {
            try
            {
                if (sound == null)
                {
                    sound = new Sound(bytes, Sound.FORMAT_TONE);
                    sound.setSoundListener(this);
                }

                if (state == Sound.SOUND_PLAYING)
                {
                    sound.stop();
                }

                sound.init(bytes, Sound.FORMAT_TONE);
                sound.play(1); // loop = 1

                // Set state and priority after play,
                // in case an exception is thrown.
                setState(Sound.SOUND_PLAYING);
                currentSoundPriority = priority;
            }
            catch(Exception e)
            {            // This is bullet-proofing (e.g. IllegalStateException).

                setState(Sound.SOUND_STOPPED);
                currentSoundPriority = 0;
            }
        }
    }

   public void soundStateChanged(Sound s, int event)
    {
        if ((s == sound) && (event == Sound.SOUND_STOPPED))
        {
            setState(Sound.SOUND_STOPPED);
            currentSoundPriority = 0;
        }
    }

    private synchronized void setState(int state)
    {
        this.state = state;
    }
```

6.4 Vibration and Device Lights Control

The Nokia UI API adds to MIDP the ability to control the phone's vibration feature and screen backlight; support for these features may vary in different Nokia models. Vibration might be used in a game to signal a collision or an explosion. A flashing backlight might be used for emphasis when a user completes a level.

Controlling Vibration:

```
void vibrate()
    {
        try
        {
            // Vibrate at 100% for 200 msec.
            DeviceControl.startVibra(100, 200);
        }
        catch(Exception e)
        {
            // An IllegalStateException is thrown if the device doesn't
            // have vibration support. Nothing can be done,lets just ignore it.
        }
}
```

Controlling Screen Backlight:

```
           void flashLights()
           {
               DeviceControl.flashLights(250);   // 250 msec
           }
```

7. NOKIA SMS API FOR JAVA MIDP

With Nokia SMS API for Java MIDP, not only can SMS messages be sent — they can also be received programmatically by Java applications running in the cell phone. Initially this capability will be available only in Nokia 3410 phones, and not in other Java-enabled phones from Nokia.

Some of the most interesting classes/interfaces in the API are:

- The MessageListener interface. This interface can be implemented by a class of the application to be notified of incoming messages. When an incoming message arrives, the notifyIncomingMessage() method is called. The application then retrieves the message using the receive() method of the MessageConnection.

- The MessageConnection interface. This interface defines the basic functionality for sending and receiving messages.

7.1 Sending and Receiving Messages

The application obtains an object implementing the MessageConnection from the Connector class by providing a URL-like string that identifies the address.

If the application specifies a full address that defines a recipient to the Connector, it gets a MessageConnection that works in a "client" mode. This kind of Connection can only be used for sending messages to the address specified when creating it.

The application can create a "server" mode MessageConnection by providing a URL string that includes only an identifier that specifies the messages intended to be received by this application. It can then use this MessageConnection object for receiving and sending messages.

The MessageConnection object also provides factory methods for creating Message objects for sending. For receiving messages, the MessageConnection supports an event-listener-based receive mechanism, in addition to a synchronous blocking receive() method.

Note that the methods for sending and receiving messages may throw a SecurityException. This will happen if the application does not have permission for these operations, thus SMS functions are protected in a way similar to network access (i.e., an option in the UI that the user can set). The user will be prompted for SMS sending in the same way as for network access. Also, some port numbers are restricted; applications tied to some ports are such that they may cause a security risk if MIDlets send SMS messages to those port numbers. For this reason, Nokia implementations will not allow MIDlets to send messages to those port numbers.

7.2 Sending a Text Message to an End User

The following example code sends the string "Hello World!" to an end user as a normal SMS:

```
try {
           String addr = "sms://+358401234567";
           MessageConnection conn = (MessageConnection)Connector.open(addr);
           TextMessage msg =
           (TextMessage)conn.newMessage(MessageConnection.TEXT_MESSAGE);
           msg.setPayloadText("Hello World!");
           conn.send(msg);
} catch (Exception e) {
   …
}
```

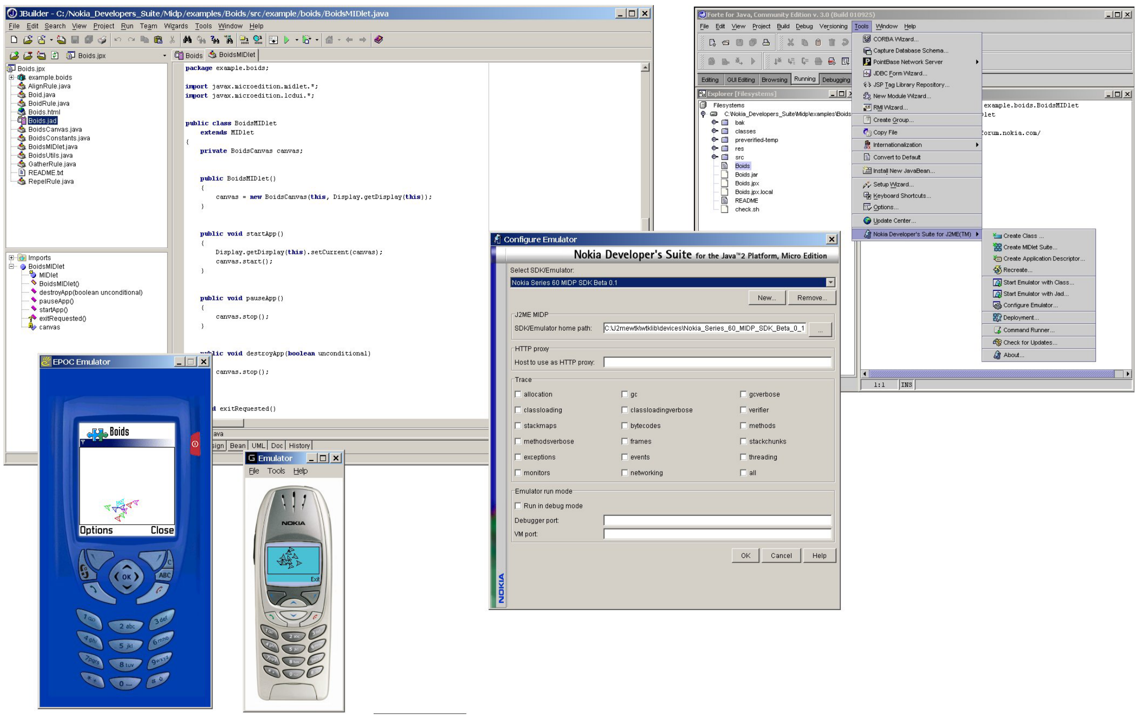
7.3 Server That Responds to Received Messages

The following example code illustrates a server application that waits for messages sent to port 5432 and responds to them:

```
try {
    String addr = "sms://:5432";
    MessageConnection conn = (MessageConnection)Connector.open(addr);
    Message msg = null;
    while (someExitCondition) {
           // wait for incoming messages
           msg = conn.receive();
           // received a message
           if (msg instanceof TextMessage) {
                      TextMessage tmsg = (TextMessage)msg;
                      String receivedText = msg.getPayloadText();
                      // respond with the same text with "Received:" inserted
                      // in the beginning
                      msg.setPayloadText("Received:" + receivedText);
                      // Note that the recipient address in the message is
                      // already correct as we are reusing the same object
                      conn.send(msg);
           }
           else {
              // Received message was not a text message, but e.g. binary
              …
           }
     }
} catch (Exception e) {
   …
}
```

8. NOKIA DEVELOPER'S SUITE FOR J2ME

Nokia Developer's Suite for J2ME provides the developer with automated code generation tools, archive builders, and software development tools, including visual MIDP phone emulators. It is possible to use Nokia Developer's Suite for J2ME in three different ways: it can be integrated with either Borland's JBuilder or Sun Microsystem's Forte for Java, or used as a standalone. Different Nokia Developer's Suite for J2ME elements are presented in Figure 19.



```java
package example.boids;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;


public class BoidsMIDlet
    extends MIDlet
{
    private BoidsCanvas canvas;


    public BoidsMIDlet()
    {
        canvas = new BoidsCanvas(this, Display.getDisplay(this));
    }


    public void startApp()
    {
        Display.getDisplay(this).setCurrent(canvas);
        canvas.start();
    }


    public void pauseApp()
    {
        canvas.stop();
    }


    public void destroyApp(boolean unconditional)

        canvas.stop();



    d exitRequested()
```

Configure Emulator

Nokia Developer's Suite for the Java™2 Platform, Micro Edition

Select SDK/Emulator: Nokia Series 60 MIDP SDK Beta 0.1

J2ME MIDP

SDK/Emulator home path: C:\J2mewtk\wtklib\devices\Nokia_Series_60_MIDP_SDK_Beta_0_1

HTTP proxy

Host to use as HTTP proxy:

Trace: ☐ allocation ☐ gc ☐ gcverbose ☐ classloading ☐ classloadingverbose ☐ verifier ☐ stackmaps ☐ bytecodes ☐ methods ☐ methodsverbose ☐ frames ☐ stackchunks ☐ exceptions ☐ events ☐ threading ☐ monitors ☐ networking ☐ all

Emulator run mode

☐ Run in debug mode

Debugger port:

VM port:

JBuilder - C:/Nokia_Developers_Suite/Midp/examples/Boids/src/example/boids/BoidsMIDlet.java
Forte for Java, Community Edition
EPOC Emulator
Boids
Options
Close
Emulator
New...
Remove...
OK
Cancel
Help
Nokia Developer's Suite for J2ME(TM)
Create Class ...
Create MIDlet Suite...
Create Application Descriptor...
Recreate...
Start Emulator with Class...
Start Emulator with Jad...
Configure Emulator...
Deployment...
Command Runner...
Check for Updates...
About...

Figure 19: Different elements of Nokia Developer's Suite for J2ME

Nokia Developer's Suite for J2ME takes advantage of JBuilder's and Forte for Java's Integrated Development Environment (IDE) services, such as project management and compiling. In addition, Nokia Developer's Suite has been built in such a way that the new feature enablers in future terminals will be easy to integrate as wireless technologies mature.

Nokia Developer's Suite for J2ME and Nokia phone emulators can be downloaded free of charge from the Forum Nokia Web site at www.forum.nokia.com. The developer's selected IDE product can be downloaded from the IDE vendor's own Web site.

Here is a list of the key features that Nokia Developer's Suite for J2ME provides today. More tools and emulators are coming, so please visit the Forum Nokia Web site for the latest updates.

Integration with

– Borland JBuilder 5,6 and 7

– Forte for Java 3, Sun ONE Studio 4

- Sun Wireless Toolkit

Non-Integrated Usage (Standalone)

Automated Wizards

– For the developer to create useful code skeletons, for example, user-defined methods

– For application packaging based on standard specifications

Interfacing with Nokia SDKs and Emulators

– Application deployment for following Nokia phone emulators

Nokia 6310i MIDP SDK

Nokia Series 40 MIDP SDK

Nokia Series 60 MIDP SDK for Symbian OS

– J2ME Wireless Toolkit interface in an IDE integration supported

Deployment to Nokia Terminals

- For Nokia Series 30 and Series 40 phones with serial cable or IrDA connection

Microsoft Windows NT users: serial cable

Microsoft Windows 2000 users: serial cable, IrDA

Additional Features

– Command Runner, a tool for creating, saving, modifying, and executing command line scripts

– Check for Updates, a tool for checking and updating a development environment, and for finding marketing and sales channels to applications

More information about Nokia Developer's Suite for J2ME can be found in the Document section of the Forum Nokia Java pages. A document called *Development Environment for J2ME™ MIDP* offers up-to-date information about Nokia J2ME/MIDP development tools. Installation guides and user's guides for the tools and emulators are also available in the Document section.

9. GAME MIDLETS

Games usually differ from other applications with regard to the greater use of graphics, but the implementer must also address these main aspects:

- Basic functionalities
- Immediate feedback

The basic functionalities common to every game are:

1. Start a new session of the game
2. Save/Pause a session
3. Retrieve the Saved/Paused session
4. Exit the current session
5. Highest scores
6. Customizing settings
7. Instructions

Feedback is essential to users during game interaction because it helps them achieve their goals. Unlike other applications, a text format is often insufficient — more immediate feedback is necessary. Consequently the MIDlet should include the use of:

- Sounds
- Vibra functionality
- Colors
- Images
- Use of backlight
- Text to show the total score achieved

The sounds and vibra features are not always appropriate to the user's environment (e.g., the user may want to play in a silent environment), therefore the settings should allow the user to switch these features on and off.

At the same time, it is important not to exaggerate with too much feedback, and to maintain it consistently across the application.

Developers must remember that if they are using special features like sound or vibration, which are not part of the standard MIDP, they should make sure that the application is working rationally, even if the feature is not supported in the user's device. The application should work on some level even if the special features are not available; at the least, it should notify the user that he/she can't use the application because the needed feature is missing. The following code shows a simple way to check if the feature is supported or not and how to respond to different situations:

```
// Vibrate at 100% for 200 msec.
            try
            {
              Class.forName("com.nokia.mid.ui.DeviceControl");
              DeviceControl.startVibra(100, 400);
            }
            catch(ClassNotFoundException e)
            {
              System.out.println(e.toString());
              //Can't use vibration because the feature
              //is not supported in the device
            }
```

10. NOKIA OK REQUIREMENTS

The Nokia OK program is a testing and evaluation process for third-party products running on Nokia mobile devices. The Nokia OK logo signifies that applications can be used with confidence because they meet the testing criteria set by Nokia. The Nokia OK program is currently available to the following Nokia J2ME-enabled phones: Nokia 6310i and Nokia 3410.

Nokia OK applications must be in line with Nokia values and therefore they must not contain references to:

- Violence against humans/animals
- Gratuitous depiction of violence against animals/nonhuman characters
- Sex, hard pornography, pedophilic imagery
- Alcohol
- Drugs and drug taking
- Use of guns/weapons (handguns, rifles, knives, axes, swords) except for fighting games based on martial arts/wrestling or fighting against "abstract" enemies (aliens, spaceships, etc.). These games must not include gratuitous depictions of the weapons' effects
- Use of knives/axes/swords/guns in human vs. human combat
- Racist imagery
- Overtly political messages, contentious political/global events, terrorism
- Religious imagery
- Gambling with money
- Stealing or robbing, e.g., bank robbery, where the player is perpetrating the crime
- Criminality, where the player is the criminal
- Swearing
- Material targeted at and marketed only to small children

Other main points are:

- The IPRs of the application should be in satisfactory condition (this will be handled in the Nokia OK agreement).
- The application must function as defined in the specification and/or user guide included with it. It also must support all the basic features that the typical user wants to perform (e.g., it must be possible to save/pause a game).
- The application must not cause any harm to system applications or data stored in the system.

The Nokia OK process is depicted in Figure 20. For more information, go to http://www.forum.nokia.com/ok:

- Nokia OK Application Guidelines for Nokia 6310i & Nokia 3410
- The Nokia OK Application Guidelines for Graphic Games on Nokia 6310i & Nokia 3410

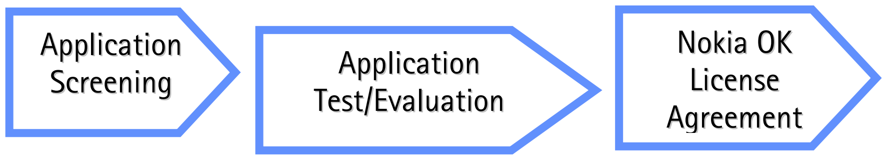


Figure 20: The Nokia OK process

11. MIDLET OTA DOWNLOADING

11.1 Downloading and Installing Java Applications to Nokia Phones

Many companies and individuals are creating Java applications for mobile phones. A typical user will download the applications into a Java-enabled device using a WAP connection. Thus, operators will play a key role, making sure that their subscribers have easy access to downloadable applications by providing WAP bookmarks to Java application portals.

For example, Club Nokia will offer downloadable Java applications to Nokia mobile phone users. All of the Nokia phones supporting J2ME/MIDP also support downloading of Java applications using the WAP browser. Many of the phone models support downloading of Java applications via PC Suite. Users of phones with PC Suite support can browse the Internet to find Java applications that they can install into their phones, or e-mail interesting applications to their friends who have phones with PC Suite.

The MIDP API specification does not specify the means for discovery and installation of MIDP applications into a phone. There are several possible ways to do this – applications can be pre-installed at the factory, downloaded over a serial cable from a PC, or downloaded Over the Air (OTA) by an application such as a WAP browser. In the future, OTA downloading is expected to be the most important way to download new MIDlets to phones. A document describing OTA provisioning is available at java.sun.com.

11.2 OTA

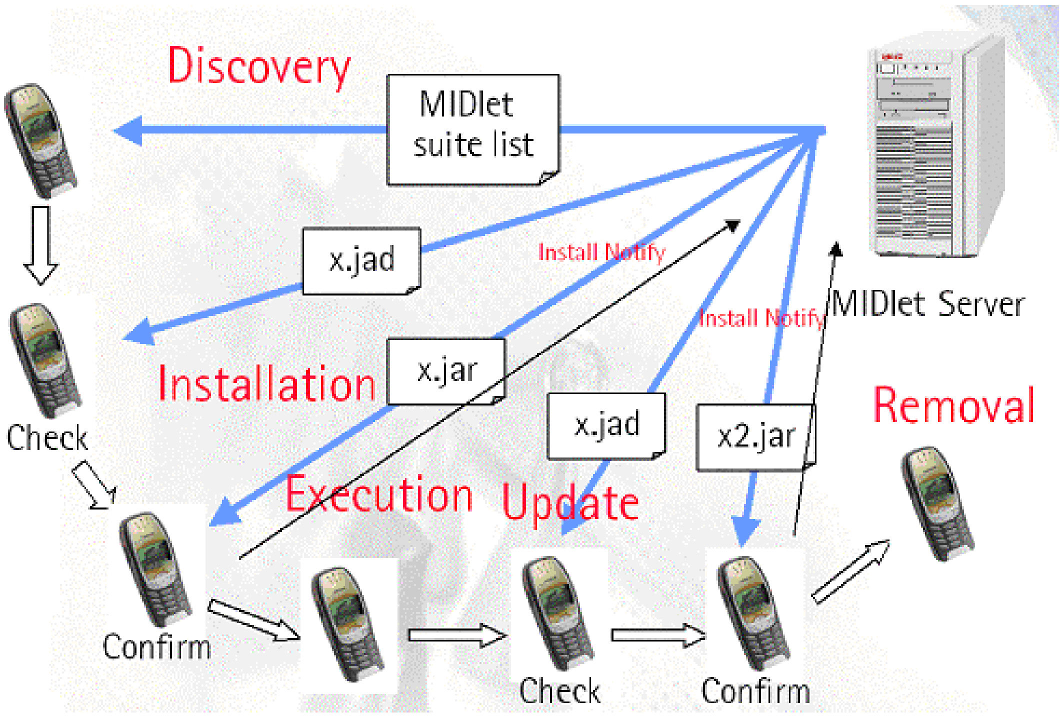


Figure 21: MIDlet deployment and lifecycle

For OTA downloads, the deployment and lifecycle of the MIDlet is roughly as follows (see Figure 21).

Discovery

A person uses a browser on his/her mobile device to browse Web pages on a remote server. The user finds a browser page containing a link to an interesting MIDP application. The link actually points at a Java Application Descriptor (*application_name.jad*). The JAD file contains the core attributes of the application. The MIME type of a JAD descriptor is: *text/vnd.sun.j2me.app-descriptor.*

Installation

When the user selects the JAD file while browsing, the browser downloads the descriptor and provides it to the Application Management Software (AMS). The AMS uses the attributes in the descriptor to check if the terminal is able to run the application. If application downloading can be performed, the AMS uses the URL given in the descriptor to locate and download the Java Archive (JAR) file from a MIDlet server.

Next, the permission to install is requested from the user; the AMS can then install the MIDlet to the terminal. At this point, the AMS may also send a confirmation notification to the server. The server uses this confirmation for billing and customer-care purposes. After successful installation, the application can be run and executed.

JAD and JAR file attributes

At a minimum, the following information must be found in the JAD file: MIDlet-name, MIDlet-vendor, MIDlet-version, MIDlet-Jar-Size, and MIDlet-Jar-URL. If the JAR file is located in the same directory as JAD, the URL address can be simply the name of the JAR. Other parameters are optional but should be used. MIDlet-name, MIDlet-vendor, and MIDlet-version must match with the manifest.mf file, which is located inside the JAR file. Other attributes may be different in the JAD and JAR manifest, in which case the JAD attribute is used.

The manifest must contain information about MicroEdition-Profile, MicroEdition-Configuration, and MIDlet-<n>. In MIDlet-<n> attribute, the <n> is the number of a particular MIDlet in the MIDlet suite. There must be a separate MIDlet-<n> attribute present for each MIDlet in the MIDlet suite. Below is an example of attributes in the JAD file and manifest file.

Attributes in JAD-file:	Explanation:
MIDlet-Name: Example	*Name of the MIDlet suite, required*
MIDlet-Vendor: Forum Nokia	*Vendor of the MIDlet, required*
MIDlet-Version: 1.0	*Version number of MIDlet, required*
MIDlet-Jar-URL: http://domain/dir/Example.jar	*Location of the JAR file, required*
MIDlet-Jar-Size: 22040	*Size of the JAR file*

Attributes in JAR-file's Manifest-file:	Explanation:
MIDlet-Name: Example	*Name of the MIDlet suite*
MIDlet-Vendor: Forum Nokia	*Vendor of the MIDlet*
MIDlet-Version: 1.0	*Version number of MIDlet*
MicroEdition-Profile: MIDP-1.0	*Version number of MID Profile*
MicroEdition-Configuration: CLDC-1.0	*Version Number of Configuration*
MIDlet-1: Example, , Example	*Name of the MIDlet-<n>, Icon, main class*

Updating

The information of the JAD file described above is needed, for example, to calculate the checksum to find out if the MIDlet has already been installed or if there is a newer version available. If an older version of the MIDlet is already installed, notification about updating can be provided.

Removal

The user can remove a MIDlet at any time and free up memory for new MIDlets.

12. LIST OF TERMS AND ABBREVIATIONS

Term or Abbreviation	Description
API	Application Programming Interface.
Canvas	A low-level UI component of LCDUI.
CLDC	Connected Limited Device Configuration. A J2ME configuration used in MIDP.
Displayable	An LCDUI object that has the capability of being placed on the display. Every LCDUI UI component that takes the whole screen is a subclass of Displayable.
Form	An LCDUI high-level UI component (Screen subclass) for presenting simple forms.
ITU-T	Telecommunication Standard.
J2ME	Java 2 Micro Edition. A Java edition that consists of different consumer electronics Java specifications and technologies. MIDP is J2ME specification.
J2SE	Java 2 Standard Edition.
JAD	Java Application Descriptor File. Contains information about the MIDlet file. JAD has a .jad file extension.
JAM	Java Application Manager. A MIDP System Component meant for managing installed and/or not installed MIDP applications in the device.
JAR	Java Archive file. A single (possibly, data compressed) file format containing the required class and resource (e.g., images, sounds) files for an application(s). It has a .jar file extension. JAR files are in ZIP file format.
KVM	Kilobyte Virtual Machine (K comes from "kilo"). It is a VM used in MIDP Reference Implementation. See also VM.
LCDUI	Limited Capability Device User Interface. A UI toolkit of MIDP.
LSK	Left Soft Key.
MIDlet	An application as specified under the MIDP specification.
MIDP	Mobile Information Device Profile. A Java profile for the J2ME. MIDP provides an open application execution environment (platform) for small, portable, and memory-constrained devices such as mobile phones.
RSK	Right Soft Key.
Screen	A subclass of Displayable. Screen is an abstract superclass for all high-level UI components in MIDP LCDUI.
VM	(Java) Virtual Machine. A software module executing Java byte code.