FORUM NOKIA

Nokia UI API Programmer's Guide

Version 1.0; June 24, 2002

NOKIA

Contents

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. Furthermore, information provided in this document is preliminary, and may be changed substantially prior to final release. 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.

The phone UI images shown in this document are for illustrative purposes and do not represent any real device.

Nokia and Nokia Connecting People 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.

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.

Nokia UI API Programmer's Guide

Version 1.0; June 24, 2002

1. Introduction

1.1 Purpose

The following Programmer's Guide describes how to use Nokia's UI API, an extension to the standard Mobile Information Device Profile (MIDP) APIs (see **[MIDP]**), which are available in Nokia's MIDP 1.0-enabled phones.

MIDP 1.0 was designed for maximum portability and concentrated only on features that could be implemented by all candidate devices. Some of those devices had no sound capabilities, hence the MIDP 1.0 specification excluded sound features. Some had very limited graphics capabilities, hence the MIDP 1.0 specification excluded some of the more advanced graphics features, such as transparency. Due to this strategy, MIDP 1.0 is portable to a wide range of devices and suitable for implementing many useful applications.

Nokia phones have sound capabilities and better graphics capabilities, so Nokia's UI API was introduced to make these features available to MIDlet developers. It is intended that such features will be covered by future versions of standard MIDP.

The following features provided by Nokia's UI API are important, especially for games:

- Support throughout the implementation for transparency in graphics (making non-rectangular sprites possible)
- The playing of simple sounds

Nokia's UI API consists of four classes and two interfaces in two new packages:

- package `com.nokia.mid.ui`
  - classes `FullCanvas`, `DirectUtils`, and `DeviceControl`
  - interface `DirectGraphics`
- package `com.nokia.mid.sound`
  - class `Sound`
  - interface `SoundListener`

1.2 References

[MIDP] http://java.sun.com/products/midp

[RTPL] Ringing Tone Parser and Generator Library
http://www.forum.nokia.com (Java section)

[SMART] Smart Messaging Specification
Revision 3.0.0
http://www.forum.nokia.com (Smart Messaging section)

2. User Interface Extensions

The Nokia UI API adds the following features:

- A full-screen Canvas class
- Drawing and filling triangles and polygons
- Drawing images reflected or rotated
- Transparency support
- Extra ways to create mutable images
- Low-level access to image pixel data

In addition, phones implementing the Nokia UI API will support transparency in the standard MIDP `Image.createImage(String)` and `Image.createImage(byte[], int, int)` methods.

2.1 Full-Screen Canvas

`[Class: com.nokia.mid.ui.FullCanvas]`

Figure 1: Boids MIDlet using Canvas (left) and FullCanvas (right)

The soft-key label area used for commands in the standard MIDP `Canvas` takes up a significant amount of the screen area, as shown on the left in Figure 1. Therefore, the Nokia UI API includes the class `FullCanvas`, which doesn't accept commands and thus uses the entire screen area for drawing. In particular, game MIDlets are likely to use `FullCanvas`.

If the phone needs to display an indicator (such as a network usage indicator), it will overwrite it on top of the `Canvas` image, usually (but not necessarily) in the top left corner. If your MIDlet does HTTP networking, you should bear this in mind when designing the MIDlet's screen layout.

`FullCanvas` does not support adding commands, and will throw an `IllegalStateException` if `addCommand` or `setCommandListener` methods are called. Instead, the soft keys will be reported using `Canvas's` keyPressed etc. methods, and `FullCanvas` defines some extra key codes for the soft keys, arrow keys, *Send* (the green phone key used for starting a call), and *End* (the red phone key used for ending a call). In many phones, pressing the *End* key will immediately terminate the MIDlet, so developers should not rely on getting key events for it.

In a well-written MIDlet, pressing any soft key will pause the action on the screen and return the user to the MIDlet's menu screen. This is preferable to emulating command labels yourself, as it is likely that emulated commands will interact differently to the phone's native commands, possibly confusing the user.

2.2 Drawing and Filling Triangles and Polygons

[Interface: com.nokia.mid.ui.DirectGraphics]

Standard MIDP 1.0 includes the ability to draw and fill rectangles, but many developers wanted the ability to draw and fill triangles and polygons, so Nokia added this capability to the Nokia UI API.

The code sample shows how to draw a simple polygon using the Nokia UI API:

```
int[] polygonX = { 30, 50, 30, 10 };
int[] polygonY = { 10, 30, 50, 30 };

void paintDiamond(Graphics g)
{
    DirectGraphics dg = DirectUtils.getDirectGraphics(g);
   dg.drawPolygon(polygonX, 0, polygonY, 0, polygonX.length, 0);
}
```

The resulting image is shown in Figure 2.

Figure 2: A simple polygon

One obvious use for filled triangles is 3-D graphics. However, with the Nokia UI API, the results are not effective on black and white screens, as the 3-D illusion is lost. On color or grayscale screens, the technique can work, but the number of triangles per second will not be great, so in practice only simple 3-D images can be drawn.

2.3 Drawing Images Reflected and Rotated

[Interface: com.nokia.mid.ui.DirectGraphics]

In MIDP 1.0 there is no way to reuse the same image for sprites moving in different directions. This often results in a JAR file containing many copies of the same image in different orientations, as shown in Figure 3.

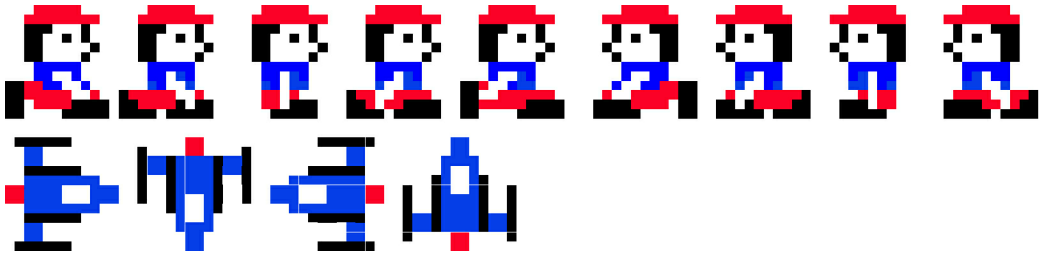

Figure 3: Sprite images in different orientations

Each animation frame of the walking man must be repeated twice and the space ship must be repeated four times. This can result in a great deal of wasted precious space in the JAR file.

Instead, you can have the image just once and use the Nokia UI API to draw the image in different orientations. To draw the man walking left, when the image file shows him walking right:

```
DirectGraphics dg = DirectUtils.getDirectGraphics(g);
dg.drawImage(img, x, y, anchor, DirectGraphics.FLIP_HORIZONTAL);
```

To draw the space ship pointing upwards, when the image file shows it pointing right:

```
DirectGraphics dg = DirectUtils.getDirectGraphics(g);
dg.drawImage(img, x, y, anchor, DirectGraphics.ROTATE_90);
```

As the code above shows, the rotation constants refer to rotation angle counter-clockwise (to rotate 90 degrees clockwise we would use `DirectGraphics.ROTATE_270`).

2.4 Transparency Support

The most significant transparency support in Nokia's UI API is not actually visible in the API definition: all phones supporting the Nokia UI API will support transparency in the standard MIDP `Image.createImage(String)` and `Image.createImage(byte[], int, int)` methods. PNG's "simple transparency" (i.e., tRNS chunk) is supported, as is the alpha channel, although alpha blending (blending drawn pixels' colors with background pixels' colors) may be unsupported.

In practice, this means that if you create your images from PNG-format files or PNG-format byte data with transparency, when you draw them, the transparent pixels will not be drawn. This allows you to have non-rectangular sprites.

Figure 4: Opaque sprites (left) and transparent sprites (right)

The images in Figure 4 clearly show the advantages of having transparent sprites: in the left image, the man's and ghost's black backgrounds block out square parts of the background image, whereas in the right image their transparent backgrounds let the background image show through and give the appearance of non-rectangular sprites. The effect is even more visible when the sprites are in motion.

2.5 Extra Ways to Create Mutable Images

[Class: com.nokia.mid.ui.DirectUtils]

In MIDP 1.0 there is only one way to create a mutable image:

```
Image img = Image.createImage(100, 100);  // create a 100x100 image
```

Unfortunately, the resulting image is full of opaque white pixels – there is no way to create an initially transparent mutable image. Since there is no way to make a pixel transparent afterwards, you can't create images containing transparency this way. In particular, you can't use this to create a transparent mutable copy of a transparent sprite image:

```
Graphics g = img.getGraphics();
g.drawImage(spriteImg, 0, 0, Graphics.TOP | Graphics.LEFT);
```

because the resulting image will be opaque white wherever the sprite image was transparent.

With the Nokia UI API, you can create a mutable image initialized to any color, including "transparent":

```
Image img = DirectUtils.createImage(100, 100, 0x00000000);
```

Here the color '0x00000000' is not RGB but ARGB – the most significant byte specifies the "alpha" value: 0xFFrrggbb is fully opaque and 0x00rrggbb is fully transparent (in which case, the value of rrggbb is irrelevant). "Alpha" can be understood as "opacity" (i.e., opaqueness – the opposite of transparency). Note that it is likely that phones won't support alpha-blending (i.e., semi-transparent lines and images blending with the background), in which case, alpha values greater than 0 are treated as fully opaque.

As well as allowing you to create transparent sprites by drawing them using Graphics and DirectGraphics methods, this is useful when you want to save space in the MIDlet's JAR file by combining all of the sprite images into one large image (thus having the per-file overhead cost only once).

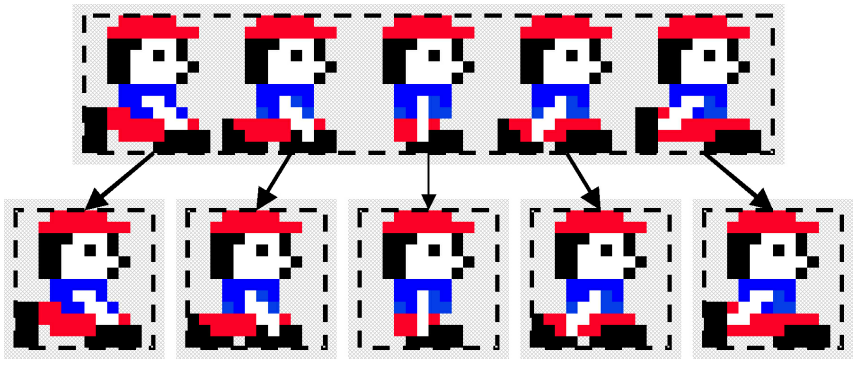

Figure 5: Splitting an image while retaining transparency

Here, you create the large image from the image file (the upper image in Figure 5), then use DirectGraphics to create transparent images for each sprite and use drawImage with suitable x and y offsets to draw the large image onto each of the small images (the lower images in Figure 5). The code is shown below (note that the individual images are 12 x 12 pixels).

```
Image fivemen = Image.createImage("/fivemen.png");
Image man[] = new Image[5];
for (int i = 0; i < 5; ++i) {
    man[i] = DirectUtils.createImage(12, 12, 0x00000000);
    Graphics g = man[i].getGraphics();
    g.drawImage(fivemen, -12 * i, 0, Graphics.TOP | Graphics.LEFT);
}
```

This will not work with MIDP 1.0, as the small images would start with opaque white backgrounds and drawing the transparent sprite images into them would result in sprites with non-transparent white backgrounds. Remember not to keep a reference to the large image afterwards, so that it can be garbage-collected.

Finally, you can create a mutable image from image byte data by:

```
Image img = DirectUtils.createImage(data, offset, length);
```

This would be useful if, for instance, you wanted to display a player's initials on the sprites for the car they are driving: you would create the sprite images, and then draw the initials into the images.

2.6 Low-Level Access to Image Pixel Data

[Interface: com.nokia.mid.ui.DirectGraphics]

Developers asked Nokia for low-level access to image pixel data so that they could get and set the data in the phone's compact internal format, rather than using, for example, PNG format. Class `DirectGraphics` includes several `getPixels` and `drawPixels` methods for doing this, but MIDlets that take full advantage of this feature will sacrifice portability even between Nokia phone models.

The following is a simple example of a method that reverses the colors of a black and white image, using the format `TYPE_BYTE_1_GRAY_VERTICAL`. All phones support getting and drawing pixels using this format, but, for example, in a color-screen phone, getting pixels in this format will produce data reduced to black and white.

```
Image createInvertedImage(Image img)
{
    int width = img.getWidth();
    int height = img.getHeight();
    Image newImage = DirectUtils.createImage(width, height
                                             0x00000000);

    int numBytes = (width * height + 7) / 8;
    byte[] pixels = new byte[numBytes];
    byte[] transparencyMask = new byte[numBytes];
    Graphics g = img.getGraphics();
    DirectGraphics dg = DirectUtils.getDirectGraphics(g);
    dg.getPixels(pixels, transparencyMask,
                 0, width, 0, 0, width, height,
                 DirectGraphics.TYPE_BYTE_1_GRAY_VERTICAL);
```

```
        for (int i = 0; i < numBytes; ++i)
            pixels[i] = (byte)~pixels[i];   // invert

        Graphics gNew = newImage.getGraphics();
        DirectGraphics dgNew = DirectUtils.getDirectGraphics(gNew);
        dgNew.drawPixels(pixels, transparencyMask,
                         0, width, 0, 0, width, height, 0,
                         DirectGraphics.TYPE_BYTE_1_GRAY_VERTICAL);

        return newImage;
    }
```

The above method will only work with mutable images, as you cannot call `getGraphics` on an immutable image. The following method will create a mutable image from a file resource:

```
Image createMutableImage(String filename) throws IOException
    {
        InputStream is = getClass().getResourceAsStream(filename);
        // add 1 so we won't reallocate just to read the EOF
        byte[] data = new byte[is.available()+1];
        int dl = 0;
        int rl;
        while ((rl = is.read(data, dl, data.length-dl)) = -1) {
            dl += rl;
            if (dl == data.length) {  // buffer full
                byte[] newData = new byte[dl + 1024];
                System.arraycopy(data, 0, newData, 0, dl);
                data = newData;
            }
        }
        Image mutable = DirectUtils.createImage(data, 0, dl);
        return mutable;
    }
```

As another example, here is a method that uses `drawPixels` and the TYPE_USHORT_4444_ARGB pixel data format to create a color checkerboard image with a transparent hole in the center:

```
    Image makeCheckerboard(int width, int height, short fg, short bg)
    {
        short[] pixels = new short[width * height];
        for (int x = 0; x < width; ++x) {
            for (int y = 0; y < width; ++y) {
                pixels[x+width*y] =
                        (((x & 4) == 0) ^ ((y & 4) == 0)) ? fg : bg;
            }
        }
```

```
            // make hole in middle
            for (int x = width / 4; x < 3 * width / 4; ++x) {
                for (int y = height / 4; y < 3 * height / 4; ++y)
{
                    pixels[x+width*y] &= 0x0FFF;
                }
            }
            Image img = Image.createImage(width, height);   // white
            Graphics g = img.getGraphics();
            DirectGraphics dg = DirectUtils.getDirectGraphics(g);
            dg.drawPixels(pixels, true, 0, width,
                          0, 0, width, height,
                          0, DirectGraphics.TYPE_USHORT_4444_ARGB);
            return img;
        }
```

This method will only work on phones that support the TYPE_USHORT_4444_ARGB format, e.g., the Nokia 7650 phone. On other phones, an `IllegalArgumentException` will be thrown from the `drawPixels` method. If you're not sure that your MIDlet will be run on a phone that supports the format you use, you should always handle that possible exception.

3. Sound Extensions

`Class: com.nokia.mid.sound.Sound`

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

- Single notes, specified by frequency and duration
- Simple tunes, specified using the Nokia Smart Messaging ringing tone binary format (the same format used for downloading new ringing tones to Nokia phones)

3.1 Playing a Single Note

Playing a simple note can be done in the following way:

```
Sound s = new Sound(440, 1000);
s.play(1);
```

The parameters to the sound constructor are frequency (in Hertz) and duration (in milliseconds). The parameter to the play method is how many times to repeat the sound (value 0 means repeat forever). The play method does not block, but returns immediately so that the sound plays simultaneously as your MIDlet continues to run. If you play a new sound before the current sound has finished playing and the phone cannot play multiple concurrent sounds (`Sound.getConcurrentSoundCount()` returns 1), then the current sound is stopped immediately and the new sound starts. You can also stop a sound directly by calling its stop method.

Some phones may only be able to play frequencies corresponding to musical notes – these frequencies are listed in the JavaDoc documentation for class Sound. Not all phones can play frequencies under 440 Hz.

You need not create a new sound object each time you want to play a sound. You can call play again, or even change the sound to be played by calling `init`:

```
s.init(494, 500);
s.play(1);
```

3.2 Playing a Simple Tune

The Nokia UI API also supports playing tunes from binary music data. While some phones support extra formats such as WAV, the format initially supported and available on all models is Nokia's binary Ringing Tone Programming Language (RTPL), defined in the Nokia Smart Messaging Specification [SMART], which can be downloaded from Forum Nokia. This is also the format used for downloading new ringing tones to your phone.

RTPL defines two types of ringing tone: basic-song (named) and temporary-song (unnamed). The Nokia UI API supports use of both types. Normally downloaded ringing tones are of the basic-song type so that there is a name for the ringing tone in the phone menu; however, the Nokia UI API does not use the name and you probably will prefer to use the slightly more compact temporary-song type.

A simple approach to generating RTPL tunes is to create the binary data using Nokia PC Composer, an application included in Nokia PC Suite. If you want more flexibility and the shortest possible binary data, you'll need to study the Nokia Smart Messaging Specification and create a simple software tool to generate the binary data from some kind of human-readable data file. Sample code for this purpose can be downloaded from Forum Nokia [RTPL].

The following is an example of how to play a simple tune:

```
byte[] data = {
    (byte)0x02, (byte)0x4A, (byte)0x3A, (byte)0x80,
    (byte)0x40, (byte)0x01, (byte)0x12, (byte)0x04,
    (byte)0x58, (byte)0x4D, (byte)0x85, (byte)0x58,
    (byte)0x59, (byte)0x86, (byte)0x18, (byte)0x69,
    (byte)0x87, (byte)0x18, (byte)0x00,
};

void playIt()
{
    Sound s = new Sound(data, Sound.FORMAT_TONE);
    s.play(1);
}
```

This will play a short scale of seven notes; typical game sound effects will be of similar lengths, but full musical tunes will be longer (after the header, each note adds 12 bits, plus, for example, 5 bits each time you change scale).

The phone must convert the binary data to an internal format in order to play it. This conversion uses a fixed-length internal buffer for the converted data, and long tunes will be truncated. The length constraint applies to the (unpublished) internal format, but for instance in the Nokia 6310i, tunes longer than about 68 notes are truncated (i.e., notes after the 68th are not played).

The following section explains one way to play longer tunes as a sequence of shorter tune parts, but there will probably be a short delay between the parts.

3.3 Sound State Model and SoundListeners

`Interface: com.nokia.mid.sound.SoundListener`

Sometimes you will want new sounds to be played after existing sounds have finished playing, rather than interrupting those sounds. This can be achieved by implementing the `SoundListener` interface and keeping a queue of sounds to play next:

```
class MyClass implements SoundListener
{
    private Vector queue = new Vector();
    private Sound sound = new Sound(0, 1);  // dummy values

    …

    MyClass()
    {
        sound.setSoundListener(this);
        …
    }


    synchronized void queueSound(byte[] data)
    {
        if (sound.getState() != Sound.SOUND_PLAYING)
        {
            sound.init(data, Sound.FORMAT_TONE);
            sound.play(1);
        }
        else
        {
            queue.addElement(data);
        }
    }

    public synchronized void soundStateChanged(Sound sound,
                                               int event)
    {
      if ((event == Sound.SOUND_STOPPED) && !queue.isEmpty())
        {
            byte[] data = (byte[])(queue.elementAt(0));
            queue.removeElementAt(0);
            sound.init(data, Sound.FORMAT_TONE);
            sound.play(1);
        }
    }
}
```

3.4 Volume Settings

When a Nokia phone is set to Silent, the Nokia UI API sound methods will not produce any sound. However, there is no way for the user to otherwise control the volume of sound produced by MIDlets, etc.

Therefore, MIDlets that produce sound may benefit from an option (e.g., in an Options screen) to set the volume level of sound produced. The sound level can be controlled using the `setGain` method of class `Sound` (*gain* means *volume level*, approximately):

```
mySound.setGain(gain);     // 0..255
```

A suitable way for the user to set the required volume level would be a standard MIDP Gauge control.

4. Vibration and Screen Backlight Control

`Class: com.nokia.mid.ui.DeviceControl`

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

4.1 Controlling Vibration

You can start the phone's vibration feature as follows:

```
DeviceControl.startVibra(100, 500);
```

The first parameter is the frequency, measured **not in Hertz** but in the range 0...100. The second parameter is the vibration duration, measured in milliseconds. The `startVibra` method does not block, but returns immediately so that the vibration happens simultaneously as your MIDlet continues to run. If you call startVibra again before the first vibration completes, the first vibration is interrupted and the second vibration starts immediately. You can interrupt a vibration directly as follows:

```
DeviceControl.stopVibra();
```

Different phones may have different vibration capabilities:

- If a phone does not support vibration, the `startVibra` method will always throw an `IllegalStateException`. Even if the phone is a model that normally supports vibration, it won't support vibration if, for instance, it is plugged into a desktop stand. The user may plug the phone into the stand while your MIDlet is running, so you should always address this possible exception.
- If the phone supports only one frequency of vibration, then frequency value 0 will result in no vibration, and values 1...100 will result in the same vibration.
- If the phone supports many frequencies of vibration, then frequency value 0 will result in no vibration, value 1 will result in the lowest frequency vibration, value 100 will result in the highest frequency vibration, and intermediate values will result in intermediate vibration frequencies.

Phones have implementation-defined limits to the duration of the vibration, and vibration will cut off at that limit even if the duration parameter was greater than the limit.

4.2 Controlling Screen Backlight

The phone's screen backlight will usually be on when the user is actively using the phone, for instance playing a game. But if it has been inactive for a few seconds, it will switch off automatically to save battery power. You can directly control this as follows:

```
DeviceControl.setLights(0, level);
```

The first parameter selects which light is affected, but currently only value 0, meaning the backlight, has any effect. The second parameter specifies the light level in the range 0...100. Value 0 means "off" (or some minimum level, if the light can't be switched completely off). Values 1...100 are mapped to the possible brightness levels of the backlight, with larger values resulting in a brighter light. Many phones only have one level, "on," so values 1...100 all mean "on." Other phones have more possible levels.

How this interacts with other backlight effects will depend on the phone. For instance, in the Nokia 6310i, if the backlight's level is set to 0 then it will still switch on briefly when keys are pressed; if the backlight's level is set higher than 0, it will not switch off even if no keys are pressed for a while.

Use this feature with care: there is no way to know what the lighting conditions are where the user is using your MIDlet, and no way to find out what level the user had set the backlight to before you started changing it (so there is no way to restore it to that level afterwards).

4.3 Flashing Lights

In games, you might want to flash the phone's lights, leaving them afterwards in whatever state they were in previously. You can do this as follows:

```
DeviceControl.flashLights(1500);
```

The parameter defines the duration in milliseconds. The `flashLights` method does not block, but returns immediately so that the flashing happens simultaneously as your MIDlet continues to run. You can interrupt the flashing by calling `flashLights(0)`.

Note that this method does not specify which lights it flashes – it may be the phone's screen backlight, some LEDs, or even possibly nothing at all. You might want to try the effect on your target phones to see what happens.

4.4 User Options

MIDlets that use vibration and lights may benefit from an option (e.g., in an Options screen) that allows that use to be enabled or disabled. Some users may prefer to disable these features.

Build Test Sell

Developing and marketing mobile applications with Nokia

Go to Forum.Nokia.com

1

Forum.Nokia.com provides the tools and resources you need for content and application development as well as the channels for sales to operators, enterprises and consumers.

Forum.Nokia.com

Subscribe to updates

2

Stay abreast of news and developments through a subscription to our regional newsletters for Europe and Africa, the Americas and Asia. Subscribing is easy and your privacy is strictly protected.

Forum.Nokia.com/newsletters

Download tools and simulators

3

Forum.Nokia.com/tools has links to tools from Nokia and other industry leaders including Adobe, AppForge, Borland, Macromedia, Metrowerks and Sun.

Forum.Nokia.com/tools

Get technical support

4

The support area contains a library of white papers, sample code and FAQs arranged by technology. The Nokia Knowledge Network enables you to ask questions of the global developer community.

Forum.Nokia.com/support
NKN.Forum.Nokia.com

Test your application

5

The Nokia OK program provides the opportunity for your application to enjoy premium placement in Nokia's sales channels.

Forum.Nokia.com/ok

Sell your application

6

Global and regional channels get your application in front of operators and XSPs, enterprises and consumers. Go to Forum.Nokia.com/business to access all of the opportunities Nokia presents.

Forum.Nokia.com/business