## Contents

1 Introduction .............................................................................................................................................. 4  
2 ARGB Image ........................................................................................................................................ 4  
   2.1 Alpha Channel ................................................................................................................................. 5  
3 drawRGB() Method ............................................................................................................................... 6  
4 Creating ARGB Image ............................................................................................................................ 6  
5 Creating Image Object from ARGB Image .............................................................................................. 9  
6 Changing ARGB Image Pixel Values .................................................................................................... 10  
   6.1 Main MIDlet Class ............................................................................................................................ 11  
   6.2 Creating and Drawing the Image ...................................................................................................... 12  
7 Summary .............................................................................................................................................. 15
## Change History

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>May 2003</td>
<td>V1.0</td>
<td>Initial document release</td>
</tr>
</tbody>
</table>
Copyright © 2003 Nokia Corporation. All rights reserved.

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.

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.

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.
1 Introduction

A look under the hood of the MIDP 2.0 specification reveals that some the most important improvements are in the Game API provided in the class `javax.microedition.lcdui.game`. By using different layers it is possible to combine different sources to generate the screen output.

The following document discusses how to work with images that are stored as an array of pixels using MIDP 2.0. It will address these topics:

- How image data is stored in an array
- Opaque versus transparent pixels
- Using the `drawRGB()` method
- Creating and populating an image array
- Creating an Image object from an image array
- Manipulating individual pixels

By using MIDP 2.0’s new features, developers will greatly improve the graphical representation of games as well as networked enterprise or stand-alone applications.

2 ARGB Image

With MIDP 2.0, developers can now create and manipulate an image through an array of integers. Each entry in the array corresponds to one pixel, with each integer value broken down as follows:

- 8 bits: Alpha (A)
- 8 bits: Red (R)
- 8 bits: Green (G)
- 8 bits: Blue (B)

A visual representation, showing each bit, of one entry in the array could look like this:

```
1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 1 0 1 0 1 0 1 0
```

Each 8-bit value is commonly referred to as a channel. Ignoring the leftmost 8 bits for the moment, the above pixel has a red channel with a value of 0000 0000 (0x00), a green channel with a value of 0101 0101 (0x55), and a blue channel with a value of 1010 1010 (0xAA). With 8 bits of storage available for each channel, there are 256 hues that can be represented in each color: red, green, and blue.
2.1 Alpha Channel

An image stored as an array of pixels is often referred to as an ARGB image. RGB obviously corresponds to the red, green, and blue values of each pixel. The A signifies the Alpha channel, or series of 8 bits that represents the opacity of the pixel.

A fully opaque pixel is stated (in binary) as 1111 1111 (0xff), as in the above example, whereas a fully transparent pixel is 0000 0000 (0x00). Figure 1 illustrates the difference between opaque and transparent pixels.

Below is a code block for creating and displaying three ARGB images. (We will return to the drawRGB() method in a moment.) Developers should concentrate on how the array is populated for each image, paying close attention to the setting of the values in the Alpha channel.

```java
public void paint(Graphics g) {
    ...  
    int x = 0, y = 0;   // Where on the display to draw the image
    int width = 35, height = 35;   // Size of the image

    // Define the array for image
    if(pixelArray == null)
        pixelArray = new int[width * height];

    // Create a fully opaque image, specifying only a value for red
    for(int i = 0; i < pixelArray.length; i++)
        pixelArray[i] = 0xffff0000;
    g.drawRGB(pixelArray, 0, width, x, y, width, height, false);  
    g.drawRect(x, y, width, height); // Box the rectangle

    // Create a “half” opaque image, specifying only a value for red
    x = 35;
    y = 35;
    for(int i = 0; i < pixelArray.length; i++)
        pixelArray[i] = 0x7fff0000;
    g.drawRGB(pixelArray, 0, width, x, y, width, height, true);
    g.drawRect(x, y, width, height); // Box the rectangle

    x = 70;
    y = 70;
    // Create a fully transparent image, specifying only a value for red
    for(int i = 0; i < pixelArray.length; i++)
        pixelArray[i] = 0x00ff0000;
    g.drawRGB(pixelArray, 0, width, x, y, width, height, true);
    g.drawRect(x, y, width, height); // Box the rectangle
}
```

In Figure 1, developers can see the difference between fully opaque pixels (top left image) and fully transparent pixels (lower right image).
3 drawRGB() Method

drawRGB() is a new method in MIDP 2.0 for drawing images stored as an array of pixels.

```java
drawRGB(int[] rgbData,  // Array containing ARGB data
        int offset,  // Offset of first value where pixel data begins
        int scanlength,  // How many array entries (pixels) per row
        int x,  // x location where image will be displayed
        int y,  // y location where image will be displayed
        int width,  // Width of the image
        int height,  // Height of the image
        boolean processAlpha)  // false if all pixels are fully opaque
```

The `scanlength` and `processAlpha` parameters deserve further clarification. When rendering the image on the display, the software that shows the pixels on the device must know how many array entries to display for each row of the image. With the example below, "width" is set to 35, indicating that there are 35 pixels in each row.

```java
int width = 35, height = 35;

// Create a fully opaque image, specifying only a value for red
for(int i = 0; i < pixelArray.length; i++)
    pixelArray[i] = 0xffff0000;
g.drawRGB(pixelArray, 0, width, x, y, width, height, false);
```

The `processAlpha` parameter is obsolete when all pixels in the array are fully opaque, but for better readability of the code it may be set to false to indicate that we do not deal with Alpha channel processing in this case (see the code above and the upper left image in Figure 1).

If any pixel in the image array is set to some value other than fully opaque, `processAlpha` may be set to true to activate the transparency. Otherwise, the Alpha channel information is ignored. This gives developers the ability to activate and deactivate transparency for images that contain transparent pixels.

```java
g.drawRGB(pixelArray, 0, width, x, y, width, height, true);
```

4 Creating ARGB Image

Let's step through the process of creating and displaying an image using an array of pixels. The image will be a red ball that is 10 pixels in height and width.

Figure 2: Image as an array of pixels
Before moving on, let’s see how the image appears without alpha processing. Figure 3 shows the image as fully opaque. The black pixels as well as the red are displayed.

Figure 3: Fully opaque image

A more aesthetically pleasing effect can be achieved when the image specifies the black pixels as transparent – the image will now appear as shown in Figure 4.

Figure 4: Image with transparent pixels

Defining the image array is simply a matter of specifying the image size in rows and columns:
// Define the array for image
int ballWidth = 10, ballHeight = 10;
if(pixelArray == null)
    pixelArray = new int[ballWidth * ballHeight];

Each row of the image is defined below. Developers should notice which pixels are set to transparent and which are opaque.

// Top row, transparent
int i = 0;
int j = 0;
for(; i < ballWidth; i++)
    pixelArray[i] = 0x00000000; // Black, transparent

// Second row
for(j = 0; j < 3; i++, j++)
    pixelArray[i] = 0x00000000; // Black, transparent
    pixelArray[i++] = 0xffff0000; // 255, 0, 0
    pixelArray[i++] = 0xffd30000; // 211, 0, 0
    pixelArray[i++] = 0xfffff000; // 255, 0, 0
    pixelArray[i++] = 0xffdb0000; // 219, 0, 0
    pixelArray[i++] = 0xff870000; // 135, 0, 0
    pixelArray[i++] = 0x00000000; // Black, transparent

// Third row
for(j = 0; j < 2; i++, j++)
    pixelArray[i] = 0x00000000; // Black, transparent
    pixelArray[i++] = 0xffff0000; // 255, 0, 0
    pixelArray[i++] = 0xfffff000; // 255, 0, 0
    pixelArray[i++] = 0x00000000; // Black, transparent

// Fourth row
for(j = 0; j < 7; i++, j++)
    pixelArray[i] = 0xffff0000; // 255, 0, 0
    pixelArray[i++] = 0xffff0000; // 255, 0, 0
    pixelArray[i++] = 0x00000000; // Black, transparent

// Fifth row
for(j = 0; j < 7; i++, j++)
    pixelArray[i] = 0xffff0000; // 255, 0, 0
    pixelArray[i++] = 0xffff0000; // 255, 0, 0
    pixelArray[i++] = 0xffd30000; // 211, 0, 0

// Sixth row
for(j = 0; j < 9; i++, j++)
    pixelArray[i] = 0xffff0000; // 255, 0, 0

// Seventh row
for(j = 0; j < 7; i++, j++)
    pixelArray[i] = 0xffff0000; // 255, 0, 0
To display the image, developers should call `drawRGB()` inside the `paint()` method:

```java
private int ballX = 50, ballY = 80;
...

public void paint(Graphics g)
{
  ...
  g.drawRGB(pixelArray, 0, ballWidth, ballX, ballY, ballWidth, ballHeight, true);
  ...
}
```

**Note:** Although we have defined various pixels as transparent, developers can specify that Alpha processing is not required. This is as simple as calling `drawRGB()` and setting the `processAlpha` parameter to `false`:

```java
g.drawRGB(pixelArray, 0, ballWidth, ballX, ballY, ballWidth, ballHeight, false);
```

All pixels will be considered fully opaque, with the resulting image identical to that shown Figure 3.

## 5 Creating Image Object from ARGB Image

Once the image has been defined as an array, developers can create an Image object and use this object in any place that accepts an Image:

```java
int ballWidth = 10, ballHeight = 10;

// Define the array for image
if(pixelArray == null)
  pixelArray = new int[ballWidth * ballHeight];
```
// Populate each row. Top row, transparent
int i = 0, j = 0;
for(; i < ballWidth; i++)
    pixelArray[i] = 0x00000000; // Black, transparent
...

// Create an Image object from the ARGB image
Image rgbImage;
int ballX = 50, ballY = 80;
rgbImage = Image.createRGBImage(pixelArray, ballWidth, ballHeight, true);
g.drawImage(rgbImage, 0, 0, Graphics.LEFT | Graphics.TOP);

One example of where a developer might choose to convert an ARGB array to an Image object is for use within other components. For example, a ChoiceGroup or List can display images next to each entry. However, this is only possible if the image is an Image object.

6 Changing ARGB Image Pixel Values

One of the benefits of having an image stored as an array of pixels is the ability to modify the image. The following example illustrates an interesting way to put this concept to use.

The basic premise of the MIDlet is to display an image (picture), as shown in Figure 5. The image is then overlaid with an ARGB image. Each pixel in the array will be tagged as fully opaque (no transparency). Both images will be written to the display, with the picture being drawn first.

Figure 5 shows the background picture; this image is then covered with the ARGB image, shown in Figure 6. When the MIDlet is started, the display in Figure 6 is what will be visible.

Using a Timer and corresponding Task, at a specified interval the ARGB image will randomly set a range of pixels to fully transparent, allowing what is displayed underneath the pixels to become visible. With each tick of the timer and update of the display, the background image will start to appear.
Figure 7: Setting ARGB pixels to transparent

When a key press is detected, the timer is canceled and all pixels in the ARGB image are set to fully transparent, revealing the background image in full (refer back to Figure 5).

6.1 Main MIDlet Class

```java
/*-----------------------------------------------*/
* DrawRGB_MIDlet
*-----------------------------------------------*/
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class DrawRGB_MIDlet extends MIDlet
{
    private Display display;
    private DrawRGB_Canvas rgbImage;

    /*-----------------------------------------------*/
```
6.2 Creating and Drawing the Image

```java
/*-----------------------------------------------*/
* DrawRGB_Canvas
*-----------------------------------------------*/
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
import java.util.*;

public class DrawRGB_Canvas extends Canvas implements CommandListener {
    private int[] pixelArray; // Image as pixel array
    private Image image = null; // Background image
    private int boxWidth = 127, boxHeight = 167, // The size of the image
        pixelArraySize, // Pixels in the array
        displayWidth, displayHeight, // Size of the display
        randomPixel; // Random pixel in the image
    private Command cmExit; // Exit midlet
    private Random random; // Random number
    private Timer tm; // Timer
    private TimerTask tt; // Task to run
    private boolean initComplete = false; // Has display been initialized
    DrawRGB_MIDlet midlet; // Main midlet

    /*-----------------------------------------------*/
* Constructor
*-----------------------------------------------*/
public DrawRGB_Canvas (DrawRGB_MIDlet midlet)
{  
    // Create instance of the canvas
    rgbImage = new DrawRGB_Canvas(this);
    display = Display.getDisplay(this);
}
```
this.midlet = midlet;

// Commands
cmExit = new Command("Exit", Command.EXIT, 0);
addCommand(cmExit);
setCommandListener(this);

//
displayWidth = getWidth();
displayHeight = getHeight();
pixelArraySize = boxWidth * boxHeight;

// Initialize random value
random = new java.util.Random();

// Read the background image from resource
try
{
    // Create immutable image
    image = Image.createImage("/seagull.png");
}
catch (java.io.IOException e)
{
    System.err.println("Unable to locate or read .png file");
}

// Create box that overlays the background image
createBox();

// Create a timer that fires off every millisecond
tm = new Timer();
tt = new DrawTask();
tm.scheduleAtFixedRate(tt, 0, 1);
}

/****************************************************************************
* Create the "box" that will cover the
* background image. A random color is selected
*---------------------------------------------------*/
private void createBox()
{
    int red, green, blue, alpha;
    long tmp;

// Define the array for image and populate
if (pixelArray == null)
{
    pixelArray = new int[pixelArraySize];

// Use random value for each color
red = (random.nextInt() >>> 1) % 256;
green = (random.nextInt() >>> 1) % 256;
blue = (random.nextInt() >>> 1) % 256;
alpha = 0xff; // Opaque

// Create single value from red, green, blue and alpha
tmp = (alpha << 24) | (red << 16) | (green << 8) | blue;

// Create image that is opaque
for (int i = 0; i < pixelArraySize; i++)
pixelArray[i] = (int) tmp;
}
}

/*--------------------------------------------------
* Call on each tick of the timer to set a range
* of pixels to transparent
*--------------------------------------------------*/
private void updateDisplay()
{
    // Get a pixel somewhere within the range of the image
    randomPixel = (random.nextInt() >> 1) % (pixelArraySize);

    // Set 50 pixels to transparent
    for (int i = randomPixel, j = 0; j < 50 && i < (pixelArraySize); j++, i++)
        pixelArray[i] = 0x00000000;
}

/*--------------------------------------------------
* Paint the images
*--------------------------------------------------*/
public void paint(Graphics g)
{
    if (initComplete == false)
    {
        // Set background to white.
        g.setColor(255, 255, 255);
        g.fillRect(0, 0, displayWidth, displayHeight);
        initComplete = true;
    }

    // Draw background image
    if (image != null)
        g.drawImage(image, 0, 0, Graphics.LEFT | Graphics.TOP);

    // Draw image stored in pixel array
    g.drawRGB(pixelArray, 0, boxWidth, 0, boxWidth, boxHeight, true);
}

/*--------------------------------------------------
* Upon keystroke, stop timer and show background image
*--------------------------------------------------*/
protected void keyPressed(int keyCode)
{
    tt.cancel();

    // Show background image by setting all pixels
    // in the ARGB image to transparent
    for (int i = 0; i < (pixelArraySize); i++)
        pixelArray[i] = 0x00000000;

    repaint();
}

/*--------------------------------------------------
* DrawTask Class
*--------------------------------------------------*/
private class DrawTask extends TimerTask
{
public final void run() {
    updateDisplay();
    repaint();
}

Note: The image size for this example was declared as:

private int boxWidth = 127, boxHeight = 167;

Although the emulators in Sun Microsystems' Wireless Toolkit (Beta 2) support images of this size, the actual image and above variables may need to be changed to support the developer's target emulator and/or device.

7 Summary

Working with images that are stored as an array of pixels adds a great deal of flexibility to image handling in MIDP. Here we've covered all the basics needed to create, manipulate, and display ARGB images. With MIDP 2.0, any application in need of an appealing graphical output can exploit layers and the alpha channel to produce results that will attract customers to an application in a way that was previously not possible.
Build > Test > Sell

Developing and marketing mobile applications with Nokia

Get started
Use free resources available through www.forum.nokia.com: articles covering the latest technical and business issues, tools for developing and deploying applications and services, bi-weekly newsletters, and active discussion boards.

www.forum.nokia.com

Download tools
Download free SDKs, emulators, simulators, and other tools that integrate with industry-leading integrated development environments.

www.forum.nokia.com/tools

Get specifications and documents
Get comprehensive device and platform specifications. Find detailed technical documentation – tutorials, technical notes, case studies, FAQs, and more.

www.forum.nokia.com/devices
www.forum.nokia.com/documents

Get support and testing services
Access our wide range of technical support services, including fee-based online support from Nokia’s experts, case resolutions from our fee-based professional support services, and Nokia Developer Hub Services, including online and onsite access to mobile infrastructure elements, such as servers and messaging centers.

www.forum.nokia.com/support

Take your applications to market
Take your applications and services to market through Nokia Tradepoint and Nokia Software Market. Other opportunities are available through Nokia Mobile Phones and other Series 60 licensees.

www.forum.nokia.com/business

Build your business with Nokia
Nokia works with selected leading developers in the games, branded media and content, and enterprise software industries. Submit your application to Nokia at www.forum.nokia.com/business.

www.forum.nokia.com/business