

GUIDEBEE
Graphics 2D API Developer Guide
On Java ME platform



James Shen
www.guidebee.biz
Guidebee Biz.

ISSUE/AMENDMENT STATUS

Issue	Date	Description	Author
1.0	08 th Oct 2007	First Version	James Shen jing.shen@guidebee.biz

TABLE OF CONTENTS

ISSUE/AMENDMENT STATUS	2
1.0 OVERVIEW OF GUIDE BEE GRAPHICS 2D JAVA ME API	5
1.1 PACKAGE DRAWING.....	6
1.2 PACKAGE GEOMETRY	7
1.3 COORDINATES	9
1.4 GRAPHICS 2D RENDERING.....	9
1.5 GEOMETRIC PRIMITIVES	11
1.5.1 <i>Points</i>	11
1.5.2 <i>Lines</i>	11
1.5.3 <i>Rectangular Shapes</i>	11
1.5.4 <i>Quadratic and Cubic Curves</i>	11
1.5.5 <i>Arbitrary Shapes</i>	12
1.5.6 <i>Areas</i>	12
1.6 TEXT	12
1.7 FONTS.....	13
1.8 TEXT LAYOUT	13
1.9 IMAGES.....	13
1.10 PEN.....	13
1.11 BRUSH.....	14
2.0 GETTING STARTED WITH GRAPHICS2D	15
3.0 WORKING WITH GEOMETRY	17
3.1 DRAWING GEOMETRIC PRIMITIVES	17
3.1.1 <i>Point</i>	17
3.1.2 <i>Line</i>	17
3.1.3 <i>Curves</i>	18
3.1.4 <i>Rectangle</i>	19
3.1.5 <i>Ellipse</i>	20
3.1.6 <i>Arc</i>	20
3.1.7 <i>ShapesDemo2D</i>	21
3.2 DRAWING ARBITRARY SHAPES	24
3.3 STROKING AND FILLING GRAPHICS PRIMITIVES	25
3.4 EXAMPLES	26
3.4.1 <i>Alias</i>	26
3.4.2 <i>Colors</i>	27
3.4.3 <i>Beziers</i>	29
3.4.4 <i>Dash</i>	34
3.4.5 <i>FillRule</i>	36
3.4.6 <i>LineCap</i>	37
3.4.7 <i>LineJoin</i>	39
3.4.8 <i>Lines</i>	41
3.4.9 <i>Ovals</i>	43
3.4.10 <i>Paths</i>	44
3.4.11 <i>Polys</i>	46
3.4.12 <i>Gradients & Pattern Brush</i>	48
4.0 WORKING WITH TEXT AND FONT	52
4.1 SELECTING A FONT	52
4.2 MEASURING TEXT AND TEXT DIRECTION.....	54
4.3 GLYPHS	57
5.0 WORKING WITH IMAGES	59

5.1 READING/LOADING AN IMAGE OR AN ICON 59

5.2 DRAWING IMAGES OR ICONS 59

5.3 ALPHA BLENDING OF IMAGES 65

5.4 DRAW TRANSPARENT IMAGES 68

5.5 ZOOM IN OR ZOOM OUT IMAGES 70

6.0 ADVANCED TOPICS IN JAVA2D..... 72

6.1 TRANSFORMING SHAPES, TEXT, AND IMAGES 72

6.2 CONSTRUCTING COMPLEX SHAPES FROM GEOMETRY PRIMITIVES..... 78

6.3 HIT PATH TEST 81

7.0 ABOUT GUIDELEE GRAPHICS 2D API 86

8.0 REFERENCES 89

1.0 Overview of Guidebee Graphics 2D Java ME API

Guidebee Graphics 2D Java ME API implements a mobile 2D graphics engine for J2ME platform (CLDC/MIDP). It handles basic shapes, paths, texts, outlined fonts and images in a uniform way.

Generally speaking, Guidebee graphics 2D API provides similar functionality with corresponding graphics 2D API on Java Standard platform.

It includes 2 packages:

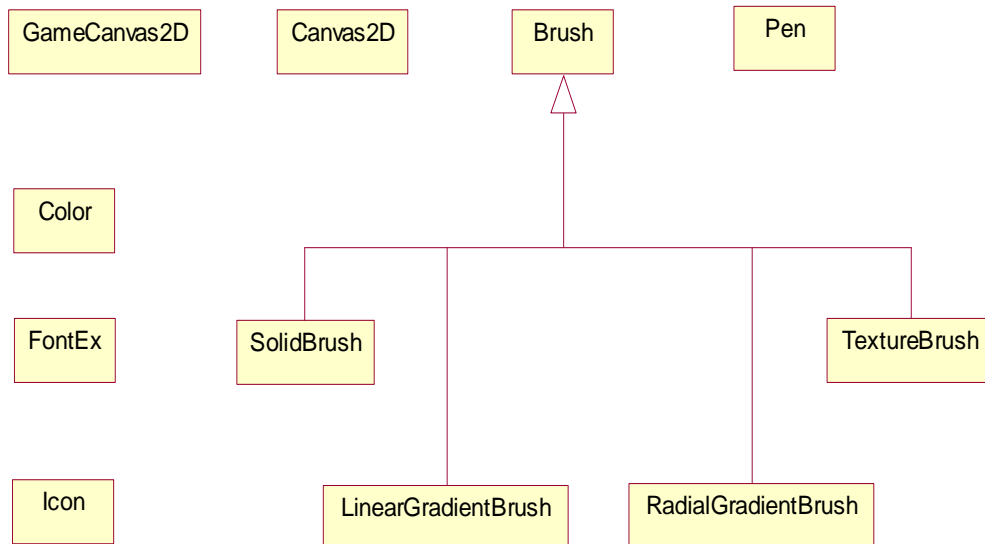
- biz.guidebee.drawing
- biz.guidebee.geometry

And it provides following capabilities:

- A wide range of geometric primitives, such as curves, rectangles, and ellipses, as well as a mechanism for rendering virtually any geometric shape
- Mechanisms for performing hit detection on shapes, text, and images
- Enhanced color support that facilitates color management
- Control of the quality of the rendering.
- Fill, stroke and dash
- Affine transformations
- Outline fonts
- Left-to-right, right-to-left and vertical text layouts
- Anti-aliasing
- Opacity
- Icon ,images

1.1 Package drawing

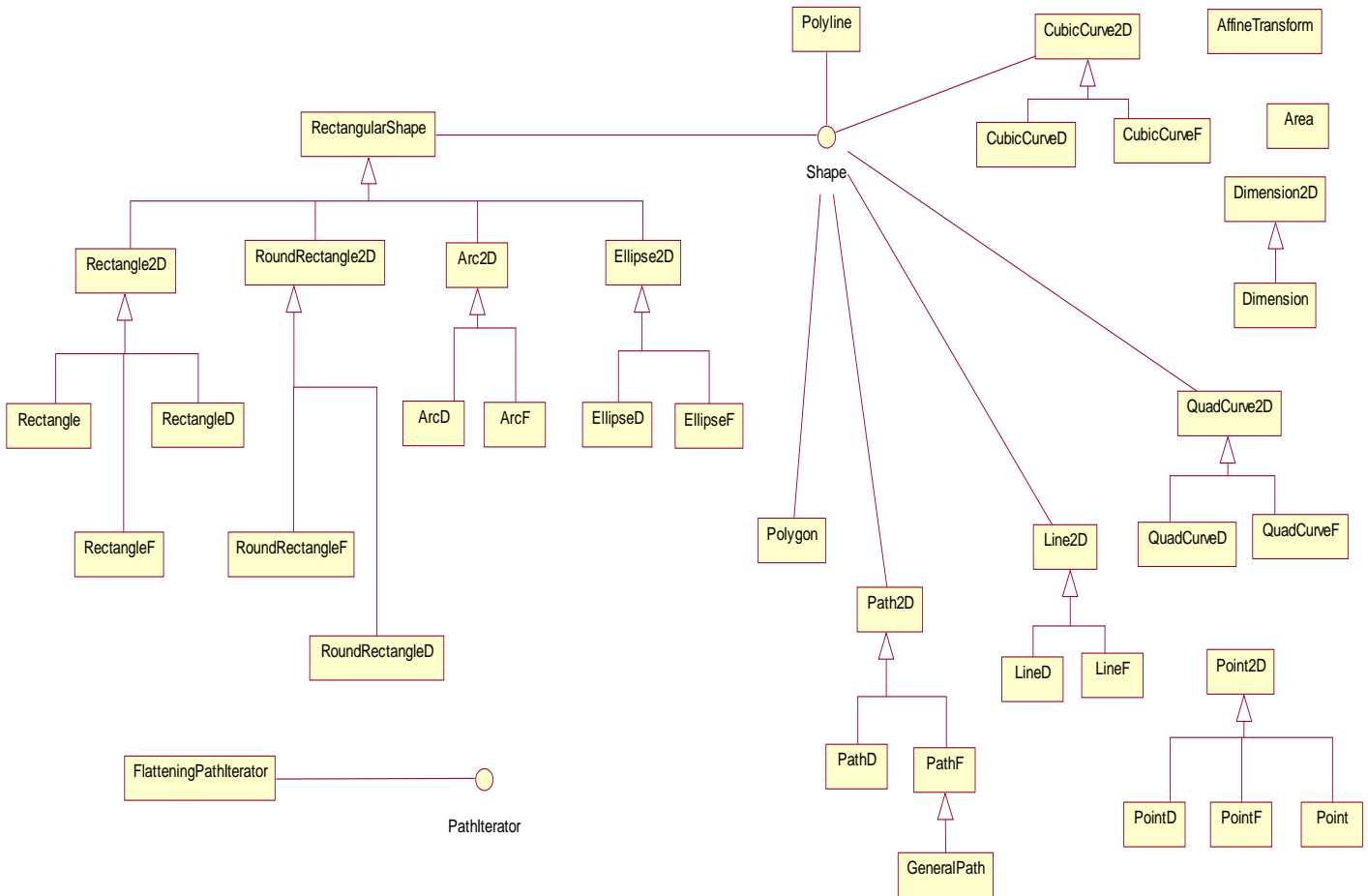
The Drawing package provides access to basic 2D graphics functionality.



Brush	Classes derived from this abstract base class define objects used to fill the interiors of graphical shapes such as rectangles, ellipses, pies, polygons, and paths.
Canvas2D	Canvas for 2D drawing.
Color	The Color class is used to encapsulate colors in the default sRGB color space Every color has an implicit alpha value of 1.0 or an explicit one provided in the constructor.
FontEx	The FontEx class represents fonts, which are used to render text in a visible way.
GameCanvas2D	GameCanvas for 2D drawing.
Graphics2D	This Graphics2D class provides more sophisticated control over geometry, coordinate transformations, color management, and text layout.
Icon	Represents a Windows icon, which is a small bitmap image used to represent an object.
LinearGradientBrush	The LinearGradientBrush class provides a way to fill a Shape with a linear color gradient pattern.
Pen	The Pen class defines a basic set of rendering attributes for the outlines of graphics primitives, which are rendered with a Graphics2D object that has its Stroke attribute set to this Pen.
RadialGradientBrush	The RadialGradientBrush class provides a way to fill a shape with a circular radial color gradient pattern.
SolidBrush	Defines a brush of a single color.
TextureBrush	The TextureBrush class provides a way to fill a Shape with a texture that is specified as an Image.

1.2 Package geometry

Package geometry provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.



PathIterator	The PathIterator interface provides the mechanism for objects that implement the Shape interface to return the geometry of their boundary by allowing a caller to retrieve the path of that boundary a segment at a time.
Shape	The Shape interface provides definitions for objects that represent some form of geometric shape.
AffineTransform	The AffineTransform class represents a 2D affine transform that performs a linear mapping from 2D coordinates to other 2D coordinates that preserves the "straightness" and "parallelness" of lines.
Arc2D	Arc2D is the abstract superclass for all objects that store a 2D arc defined by a framing rectangle, start angle, angular extent (length of the arc), and a closure type (OPEN, CHORD, or PIE).
ArcD	This class defines an arc specified in double precision.
ArcF	This class defines an arc specified in float precision.

Area	An Area object stores and manipulates a resolution-independent description of an enclosed area of 2-dimensional space.
CubicCurve2D	The CubicCurve2D class defines a cubic parametric curve segment in (x, y) coordinate space.
CubicCurveD	A cubic parametric curve segment specified with double coordinates.
CubicCurveF	A cubic parametric curve segment specified with float coordinates.
Dimension	The Dimension class encapsulates the width and height of a component (in integer precision) in a single object.
Dimension2D	The Dimension2D class is to encapsulate a width and a height dimension.
Ellipse2D	The Ellipse2D class describes an ellipse that is defined by a framing rectangle.
EllipseD	The EllipseD class defines an ellipse specified in double precision.
EllipseF	The EllipseF class defines an ellipse specified in float precision.
FlatteningPathIterator	The FlatteningPathIterator class returns a flattened view of another PathIterator object.
GeneralPath	The GeneralPath class represents a geometric path constructed from straight lines, and quadratic and cubic (Bezier) curves.
Line2D	This Line2D represents a line segment in (x, y) coordinate space.
LineD	A line segment specified with double coordinates.
LineF	A line segment specified with float coordinates.
Path2D	The Path2D class provides a simple, yet flexible shape which represents an arbitrary geometric path.
PathD	The PathD class defines a geometric path with coordinates stored in double precision floating point.
PathF	The PathF class defines a geometric path with coordinates stored in single precision floating point.
Point	A point representing a location in (x, y) coordinates space, specified in integer precision.
Point2D	The Point2D class defines a point representing a location in (x, y) coordinate space.
PointD	The PointD class defines a point specified in double precision.
PointF	The PointF class defines a point specified in float precision.
Polygon	The Polygon class encapsulates a description of a closed, two-dimensional region within a coordinate space.
Polyline	The Polyline class encapsulates a description of a collection of line segments within a coordinate space.
QuadCurve2D	The QuadCurve2D class defines a quadratic parametric curve segment in (x, y) coordinate space.
QuadCurveD	A quadratic parametric curve segment specified with double coordinates.
QuadCurveF	A quadratic parametric curve segment specified with float coordinates.
Rectangle	A Rectangle specifies an area in a coordinate space that is enclosed by the Rectangle object's upper-left point (x, y) in the coordinate space, its width, and its height.
Rectangle2D	The Rectangle2D class describes a rectangle defined by a location (x, y) and dimension (w x h).
RectangleD	The RectangleD class defines a rectangle specified in double coordinates.

RectangleF	The RectangleF class defines a rectangle specified in float coordinates.
RectangularShape	RectangularShape is the base class for a number of Shape objects whose geometry is defined by a rectangular frame.
RoundRectangle2D	The RoundRectangle2D class defines a rectangle with rounded corners defined by a location (x, y), a dimension (w x h), and the width and height of an arc with which to round the corners.
RoundRectangleD	The RoundRectangleD class defines a rectangle with rounded corners all specified in double coordinates.
RoundRectangleF	The RoundRectangleF class defines a rectangle with rounded corners all specified in float coordinates.

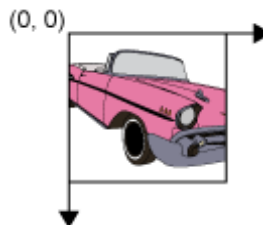
1.3 Coordinates

The Graphics 2D API maintains two coordinate spaces.

- User space – The space in which graphics primitives are specified
- Device space – The coordinate system of an output device such as a screen.

User space is a device-independent logical coordinate system, the coordinate space that your program uses. All geometries passed into Graphics 2D rendering routines are specified in user-space coordinates.

When the default transformation from user space to device space is used, the origin of user space is the upper-left corner of the component’s drawing area. The x coordinate increases to the right and the y coordinate increases downward, as shown in the following figure. The top-left corner of a window is 0, 0. All coordinates are specified using integers, which is usually sufficient. However, some cases require floating point or even double precision which are also supported.



Device space is a device-dependent coordinate system that varies according to the target rendering device. The necessary conversions between user space and device space are performed automatically during rendering.

1.4 Graphics 2D Rendering

The core class of the Graphics 2D API is biz.guidebee.drawing.Graphics2D class, similar to class javax.microedition.lcdui.Graphics, but provide access to the enhanced graphics and rendering features of the Graphics 2D API. These features include:

- Rendering the outline of any geometry primitive, using the pen and brush attributes (draw method).
- Rendering any geometry primitive by filling its interior with the color or pattern specified by the brush attributes (fill method).

- Rendering any text string (drawString method). The font attribute is used to convert the string to glyphs, which are then filled with the color or pattern specified by the brush attributes.
- Rendering the specified image (drawImage method).

In addition, the Graphics2D class supports the Graphics rendering methods for particular shapes, such as drawOval and fillRect.

All methods that are represented above can be divided into two groups. First group contains methods to draw a shape. Second group contains methods to affect the rendering. The last group operates with a term attributes.

You can modify the state attributes that form the Graphics2D context for following purposes:

- To vary the stroke width (Pen)
- To change how strokes(Pen) are joined together
- To translate, rotate, scale, or shear objects when they are rendered
- To define colors and brushes to fill shapes with

To use Graphics 2D API features, you first make a new instance of Canvas2D or GameCanvas2D (the difference between Canvas2D and GameCanvas2D is their base classes, Canvas2D inherits from javax.microedition.lcudi.Canvas, while GameCanvas2D inherits from javax.microedition.lcudi.game.GameCanvas, but they provide similar Graphics APIs), then call Canvas2D's (or GameCanvas2D's) getGraphics methods. For example:

```
display = Display.getDisplay(this);
canvas = new Canvas2D(display);
graphics2D = canvas.getGraphics();
```

When you set an attribute, you pass in the appropriate attribute object. As the following example shows, to change the Brush attribute to a blue-green gradient fill, you would construct a LinearGradientBrush object and then call the setDefaultBrush method.

```
Color []colors=new Color[]{Color.BLUE,Color.GREEN};
brush=new LinearGradientBrush(50,50,150,125,fractions,
    colors,Brush.REFLECT);
graphics2D.setDefaultBrush(brush);
```

Or set the brush attributes when call one of the fillXXX () methods.

```
graphics2D.fillRectangle(brush,new Rectangle(10,75,120,80));
```

1.5 Geometric Primitives

The Graphics 2D API provides a useful set of standard shapes such as points, lines, rectangles, arcs, ellipses, and curves. Arbitrary shapes can be represented by combinations of straight geometric primitives.

The Shape interface represents a geometric shape, which has an outline and an interior. This interface provides a common set of methods for describing and inspecting two-dimensional geometric objects and supports curved line segments and multiple sub-shapes. The Shape interface can support curves segments.

1.5.1 Points

The Point2D class defines a point representing a location in (x, y) coordinate space. The term “point” in the Graphics 2D API is not the same as a pixel. A point has no area, does not contain a color, and cannot be rendered.

Points are used to create other shapes. The Point2D class also includes a method for calculating the distance between two points.

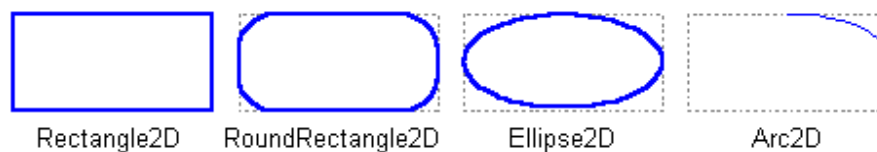
1.5.2 Lines

The Line2D class is an abstract class that represents a line. A line’s coordinates can be retrieved as double. The Line2D class includes several methods for setting a line’s endpoints.

Also, you can create a straight line segment by using the GeneralPath class described below.

1.5.3 Rectangular Shapes

The Rectangle2D, RoundRectangle2D, Arc2D, and Ellipse2D primitives are all derived from the RectangularShape class. This class defines methods for Shape objects that can be described by a rectangular bounding box. The geometry of a RectangularShape object can be extrapolated from a rectangle that completely encloses the outline of the Shape.

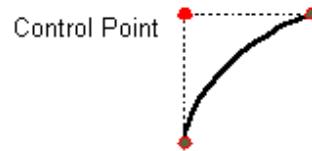


1.5.4 Quadratic and Cubic Curves

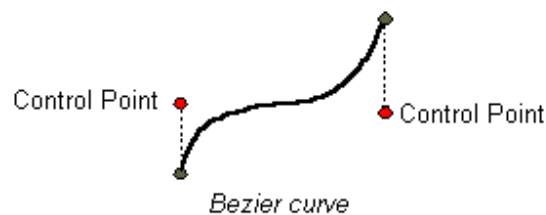
The QuadCurve2D enables you to create quadratic parametric curve segments. A quadratic curve is defined by two endpoints and one control point.

The CubicCurve2D class enables you to create cubic parametric curve segments. A cubic curve is defined by two endpoints and two control points. The following are examples of quadratic and cubic curves. See [Stroking and Filling](#) for implementations of cubic and quadratic curves.

This figure represents a quadratic curve.



This figure represents a cubic curve.



1.5.5 Arbitrary Shapes

The `GeneralPath` class enables you to construct an arbitrary shape by specifying a series of positions along the shape's boundary. These positions can be connected by line segments, quadratic curves, or cubic (Bezier) curves. The following shape can be created with three line segments and a cubic curve.



1.5.6 Areas

With the `Area` class, you can perform Boolean operations, such as union, intersection, and subtraction, on any two `Shape` objects. This technique, often referred to as constructive area geometry, enables you to quickly create complex `Shape` objects without having to describe each line segment or curve.

1.6 Text

The Graphics 2D API has various text rendering capabilities including methods for rendering strings and entire classes for setting font attributes and performing text layout.

If you just want to draw a static text string, the most direct way to render it directly through the `Graphics2D` class by using the `drawString` method. To specify the font, you use the `setDefaultFont ()` method of the `Graphics2D` class or gives the `Font` instance as the parameter when calling `drawString` method.

1.7 Fonts

The shapes that a font uses to represent the characters in a string are called glyphs. A particular character or combination of characters might be represented as one or more glyphs.

A font can be thought of as a collection of glyphs. A single font might have many faces, such as italic and regular. All of the faces in a font have similar typographic features and can be recognized as members of the same family. In other words, a collection of glyphs with a particular style form a font face. A collection of font faces forms a font family. The collection of font families forms the set of fonts that are available on the system.

The Graphics2D API support two types of fonts, one is original `javax.microedition.lcdui.Font` class, the other is `biz.guidebee.drawing.FontEx`, the `FontEx` class provide more functionalities than `Font` class, it allows affine transformation like normal `Shape` object.

1.8 Text Layout

Before text can be displayed, it must be laid out so that the characters are represented by the appropriate glyphs in the proper positions.

It supports Left-to-right, right-to-left and vertical text layouts.

1.9 Images

In the Graphics 2D API an image is typically a rectangular two-dimensional array of pixels, where each pixel represents the color at that position of the image and where the dimensions represent the horizontal extent (width) and vertical extent (height) of the image as it is displayed.

The Graphics 2D API also support drawing standard window Icon and support alpha blending of the images.



1.10 Pen

The `Pen` class defines a basic set of rendering attributes for the outlines of graphics primitives, which are rendered with a `Graphics2D` object that has its `Stroke` attribute set to this `Pen`. The rendering attributes defined by `Pen` describe the shape of the mark made by a pen drawn along the outline of a `Shape` and the decorations applied at the ends and joins of path segments of the `Shape`. These rendering attributes include:

Width

The pen width, measured perpendicularly to the pen trajectory.

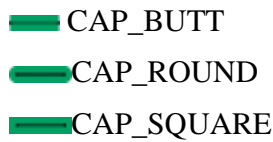
Join style

The join style is the decoration that is applied where two line segments meet. It supports the following three join styles:



End-cap style

The end-cap style is the decoration that is applied where a line segment ends. BasicStroke supports the following three end-cap styles:

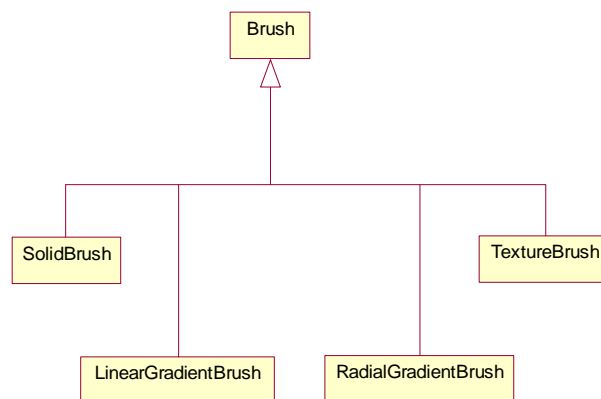


Dash attributes

The dash style defines the pattern of opaque and transparent sections applied along the length of the line. The dash style is defined by a dash array and a dash phase.

1.11 Brush

Classes derived from this abstract base class define objects used to fill the interiors of graphical shapes such as rectangles, ellipses, pies, polygons, and paths.



When fill, it can use single color (SolidBrush), Gradients (LinerGradientBrush or RadialGradientBursh) or Patterns (TextureBrush) to fill the internal part of any shape or text.

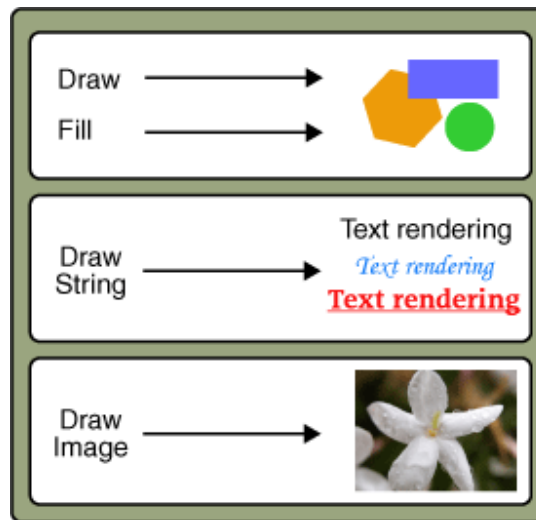
2.0 Getting Started with Graphics2D

Most methods of the Graphics2D class can be divided into two basic groups:

- Draw and fill methods, enabling you to render basic shapes, text, and images
- Attributes setting methods, which affect how that drawing and filling appears

Methods such as `setDefaultFont`, `setDefaultPen` and `setDefaultBrush` define how draw and fill methods render.

This figure illustrates how these methods relate to graphic objects:



Drawing methods include:

- `drawChars` – For drawing text

```
FontEx font=FontEx.getSystemFont();
char [] longLine = "Happy,James! ".toCharArray();
graphics2D.setDefaultPen(new Pen(Color.BLACK,1));
graphics2D.drawChars(font,16,longLine,0,longLine.length,10,10);
```

- `drawImage` – For drawing images

```
graphics2D.drawImage(bi,offX,offY,0xFFFF00FF,alpha);
```

- `drawLine`, `drawArc`, `drawRect`, `drawOval`, `drawPolygon` – For drawing geometric shapes

```
Pen pen=new Pen(greenColor,1);
graphics2D.drawLine(pen,20,150,60,50);
```

Depending on your current need, you can choose one of several methods in the Graphics class based on the following criteria:

- Whether you want to render the image at the specified location in its original size or scale it to fit inside the given rectangle

- Whether you prefer to fill the transparent areas of the image with color or keep them transparent

Fill methods apply to geometric shapes and include fillArc, fillRect, fillOval, fillPolygon.

Whether you draw a line of text or an image, remember that in 2D graphics every point is determined by its x and y coordinates. All of the draw and fill methods need this information which determines where the text or image should be rendered.

3.0 Working with Geometry

This section shows you how to use the Graphics2D class to draw graphic primitives as well as arbitrary shapes, and how to display graphics with fancy outline and fill styles.

3.1 Drawing Geometric Primitives

The Graphics2D API provides several classes that define common geometric objects such as points, lines, curves, and rectangles.

The PathIterator interface defines methods for retrieving elements from a path.

The Shape interface provides a set of methods for describing and inspecting geometric path objects. This interface is implemented by the GeneralPath class and other geometry classes.

3.1.1 Point

The Point class creates a point representing a location in (x, y) coordinate space. The subclasses PointF and PointD provide correspondingly float and double precision for storing the coordinates of the point.

```
//Create Point2D
Point2D point = new Point2D (x, y);
```

To create a point with the coordinates 0, 0 you use the default constructor, PointD().

You can use the setLocation method to set the position of the point as follows:

- setLocation(double x, double y) – To set the location of the point- defining coordinates as double values
- setLocation(Point2D p) – To set the location of the point using the coordinates of another point.

Also, the Point2D class has methods to calculate the distance between the current point and a point with given coordinates, or the distance between two points.

3.1.2 Line

The Line2D class represents a line segment in (x, y) coordinate space. The LineF and LineD subclasses specify lines in float and double precision. For example:

```
// draw Line2D.Double
graphics2D.draw (pen, new LineD(x1, y1, x2, y2));
```



This class includes several setLine() methods to define the endpoints of the line.

Alternatively, the endpoints of the line could be specified by using the constructor for the LineF class as follows:

- LineF(float X1, float Y1, float X2, float Y2)
- LineF(Point2D p1, Point2D p2)

The Pen object in the Graphics2D class defines the stroke for the line path.

3.1.3 Curves

The Graphics 2D API enables you to create a quadratic or cubic curve segment.

3.1.3.1 Quadratic Curve Segment

The QuadCurve2D class implements the Shape interface. This class represents a quadratic parametric curve segment in (x, y) coordinate space. The QuadCurveF and QuadCurveD subclasses specify a quadratic curve in float and double precision.

Several setCurve methods are used to specify two endpoints and a control point of the curve, whose coordinates can be defined directly, by the coordinates of other points and by using a given array.

A very useful method, setCurve(QuadCurve2D c), sets the quadratic curve with the same endpoints and the control point as a supplied curve. For example:

```
// create new QuadCurve2D.Float
QuadCurve2D q = new QuadCurveF();
// draw QuadCurve2D.Float with set coordinates
graphics2D.draw(pen,q.setCurve(x1, y1, ctrlx, ctrly, x2,
y2));
```



3.1.3.2 Cubic Curve Segment

The CubicCurve2D class also implements the Shape interface. This class represents a cubic parametric curve segment in (x, y) coordinate space. CubicCurveF and CubicCurveD subclasses specify a cubic curve in float and double precision.

The CubicCurve2D class has similar methods for setting the curve as the QuadraticCurve2D class, except with a second control point. For example:

```
// create new CubicCurve2D.Double
CubicCurve2D c = new CubicCurveD();
// draw CubicCurve2D.Double with set coordinates
graphics2D.draw(pen, c.setCurve(x1, y1, ctrlx1,
ctrlx2, ctrly1, ctrly2, x2, y2));
```



3.1.4 Rectangle

Classes that specify primitives represented in the following example extend the `RectangularShape` class, which implements the `Shape` interface and adds a few methods of its own.

These methods enable you to get information about a shape's location and size, to examine the center point of a rectangle, and to set the bounds of the shape.

The `Rectangle2D` class represents a rectangle defined by a location (x, y) and dimension ($w \times h$). The `RectangleF` and `RectangleD` subclasses specify a rectangle in float and double precision. For example:

```
// draw Rectangled
graphics2D.draw(pen,new Rectangled(x, y,
                                rectwidth,
                                rectheight));
```



The `RoundRectangle2D` class represents a rectangle with rounded corners defined by a location (x, y), a dimension ($w \times h$), and the width and height of the corner arc. The `RoundRectangleF` and `RoundRectangleD` subclasses specify a round rectangle in float and double precision.

The rounded rectangle is specified with following parameters:

- Location
- Width
- Height
- Width of the corner arc
- Height of the corner arc

To set the location, size, and arcs of a `RoundRectangle2D` object, use the method `setRoundRect(double a, double y, double w, double h, double arcWidth, double arcHeight)`. For example:

```
// draw RoundRectangled
graphics2D.draw(pen,new RoundRectangled(x, y,
                                        rectwidth,
                                        rectheight,
                                        10, 10));
```

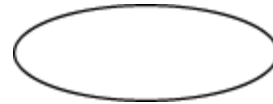


3.1.5 Ellipse

The Ellipse2D class represents an ellipse defined by a bounding rectangle. The EllipseF and EllipseD subclasses specify an ellipse in float and double precision.

Ellipse is fully defined by a location, a width and a height. For example:

```
// draw EllipseD
graphics2D.draw(pen, new EllipseD(x, y,
                                rectwidth,
                                rectheight));
```



3.1.6 Arc

To draw a piece of an ellipse, you use the Arc2D class. This class represents an arc defined by a bounding rectangle, a start angle, an angular extent, and a closure type. The ArcF and ArcD subclasses specify an ellipse in float and double precision.

The Arc2D class defines the following three types of arcs, represented by corresponding constants in this class: OPEN, PIE and CHORD.



Several methods set the size and parameters of the arc:

- Directly, by coordinates
- By supplied Point2D and Dimension2D
- By copying an existing Arc2D

Also, you can use the setArcByCenter method to specify an arc from a center point, given by its coordinates and a radius.

```
// draw ArcD
graphics2D.draw(pen, new ArcD(x, y,
                              rectwidth,
                              rectheight,
                              90, 135,
                              Arc2D.OPEN));
```



3.1.7 ShapesDemo2D

The ShapesDemo2D.java code example contains implementations off all described geometric primitives.

```
//----- PACKAGE -----
package biz.guidebee.example.drawing2d;

//----- IMPORTS -----
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

import biz.guidebee.drawing.*;
import biz.guidebee.geometry.*;

//[----- MAIN CLASS -----]
//----- REVISIONS -----
// Date      Name      Tracking #      Description
// -----
// 01SEP2007 James Shen      Initial Creation
//-----
/**
 * The example shows how to draw graphics 2D primitives.
 */
public class ShapesDemo2D extends MIDlet
{
    /**
     * The display object associated with the MIDlet.
     */
    private Display display;

    /**
     * The Canvas object.
     */
    private GameCanvas2D canvas;

    /**
     * The graphics object.
     */
    private Graphics2D graphics2D;

    final static int maxCharHeight = 15;
    final static int minFontSize = 6;

    final static Color bg = Color.white;
    final static Color fg = Color.black;
    final static Color red = Color.red;
    final static Color white = Color.white;

    final static Pen pen = new Pen(fg,1);
    final static SolidBrush brush=new SolidBrush(red);
    final static Pen widePen = new Pen(fg,8);

    final static int dash1[] = {10,10};
    final static Pen dashed = new Pen(fg,1, Pen.CAP_BUTT,
                                         Pen.JOIN_MITER,
                                         10, dash1, 0);

    Dimension totalSize;

    FontEx font=FontEx.getSystemFont();

    public ShapesDemo2D() {

        display = Display.getDisplay(this);
        canvas = new GameCanvas2D(display,true,true);
        graphics2D = canvas.getGraphics2D();
    }
}
```

```

}

public void startApp() throws MIDletStateChangeException {
    display.setCurrent(canvas);
    //Clear the canvas with white color.
    graphics2D.clear(bg);
    Dimension d = new Dimension(canvas.getWidth(),canvas.getHeight());
    int gridWidth = d.width / 2;
    int gridHeight = d.height / 6;

    int x = 5;
    int y = 7;
    int rectWidth = gridWidth - 2*x;
    int stringY = gridHeight - 3 - 2;
    int rectHeight = stringY - 16 - y - 2;

    graphics2D.draw(pen,new LineD(x, y+rectHeight-1, x + rectWidth, y));

    graphics2D.drawChars(font,16,"Line2D".toCharArray(),
        0,"Line2D".toCharArray().length, x, stringY);
    x += gridWidth;

    graphics2D.draw(pen,new RectangleD(x, y, rectWidth, rectHeight));
    graphics2D.drawChars(font,16,"Rectangle2D".toCharArray(),
        0,"Rectangle2D".toCharArray().length, x, stringY);
    x += gridWidth;

    x = 5;
    y += gridHeight;
    stringY += gridHeight;

    graphics2D.draw(pen,new RoundRectangleD(x, y, rectWidth,
        rectHeight, 10, 10));
    graphics2D.drawChars(font,16,"RoundRect2D".toCharArray(),0,
        "RoundRect2D".toCharArray().length, x, stringY);

    x += gridWidth;

    graphics2D.draw(pen,new ArcD(x, y, rectWidth, rectHeight, 90,
        135, Arc2D.OPEN));
    graphics2D.drawChars(font,16,"Arc2D".toCharArray(),0,
        "Arc2D".toCharArray().length, x, stringY);

    x = 5;
    y += gridHeight;
    stringY += gridHeight;

    graphics2D.draw(pen,new EllipseD(x, y, rectWidth, rectHeight));
    graphics2D.drawChars(font,16,"Ellipse2D".toCharArray(),0,
        "Ellipse2D".toCharArray().length, x, stringY);
    x += gridWidth;

    // draw GeneralPath (polygon)
    int x1Points[] = {x, x+rectWidth, x, x+rectWidth};
    int y1Points[] = {y, y+rectHeight, y+rectHeight, y};
    GeneralPath polygon = new GeneralPath(GeneralPath.WIND_EVEN_ODD,
        x1Points.length);
    polygon.moveTo(x1Points[0], y1Points[0]);
    for ( int index = 1; index < x1Points.length; index++ ) {
        polygon.lineTo(x1Points[index], y1Points[index]);
    };
    polygon.closePath();

    graphics2D.draw(pen,polygon);
    graphics2D.drawChars(font,16,"Path".toCharArray(),0,
        "Path".toCharArray().length, x, stringY);

    x = 5;
    y += gridHeight;

```

```

stringY += gridHeight;

int x2Points[] = {x, x+rectWidth, x, x+rectWidth};
int y2Points[] = {y, y+rectHeight, y+rectHeight, y};
GeneralPath polyline = new GeneralPath(GeneralPath.WIND_EVEN_ODD,
                                       x2Points.length);
polyline.moveTo (x2Points[0], y2Points[0]);
for ( int index = 1; index < x2Points.length; index++ ) {
    polyline.lineTo(x2Points[index], y2Points[index]);
};

graphics2D.draw(pen,polyline);
graphics2D.drawChars(font,16,"Path(Open)".toCharArray(),0,
                    "Path(Open)".toCharArray().length, x, stringY);

x += gridWidth;
graphics2D.setPenAndBrush(pen,brush);
graphics2D.fill(null,new Rectangled(x, y, rectWidth, rectHeight));
graphics2D.drawChars(font,16,"Rect(Filled)".toCharArray(),0,
                    "Rect(Filled)".toCharArray().length, x, stringY);

x = 5;
y += gridHeight;
stringY += gridHeight;
Color[]colors=new Color[]{red,white};
int[]fractions=new int[]{0,255};
LinearGradientBrush redtowhite =
    new LinearGradientBrush(x,y,x+rectWidth, y,fractions,colors);
graphics2D.setPenAndBrush(pen,redtowhite);
graphics2D.fill(null,new RoundRectangled(x, y, rectWidth,
                                       rectHeight, 10, 10));
graphics2D.drawChars(font,16,"RndRect(Filled)".toCharArray(),0,
                    "RndRect(Filled)".toCharArray().length, x, stringY);
x += gridWidth;

graphics2D.setPenAndBrush(pen,brush);
graphics2D.fill(null,new ArcD(x, y, rectWidth, rectHeight, 90,
                             135, Arc2D.OPEN));

graphics2D.drawChars(font,16,"Arc2D(Filled)".toCharArray(),0,
                    "Arc2D(Filled)".toCharArray().length, x, stringY);

x = 5;
y += gridHeight;
stringY += gridHeight;
int x3Points[] = {x, x+rectWidth, x, x+rectWidth};
int y3Points[] = {y, y+rectHeight, y+rectHeight, y};
GeneralPath filledPolygon = new GeneralPath(GeneralPath.WIND_EVEN_ODD,
                                       x3Points.length);
filledPolygon.moveTo(x3Points[0], y3Points[0]);
for ( int index = 1; index < x3Points.length; index++ ) {
    filledPolygon.lineTo(x3Points[index], y3Points[index]);
};
filledPolygon.closePath();
graphics2D.setPenAndBrush(pen,brush);
graphics2D.fill(null,filledPolygon);

graphics2D.drawChars(font,16,"Filled and Stroked Path".toCharArray(),0,
                    "Filled and Stroked Path".toCharArray().length, x, stringY);

}
graphics2D.invalidate();

public void pauseApp() {
}

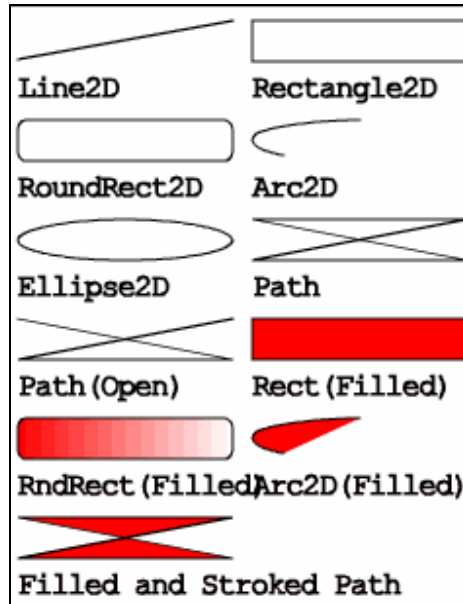
public void destroyApp(boolean unconditional)
throws MIDletStateException {
    canvas.flush();
}

```

```

        canvas = null;
        System.gc();
    }
}
    
```

What's shown on the screen will be like the following:



3.2 Drawing Arbitrary Shapes

To create more complicated geometry, such as polygons, polylines, or stars you use another class from this package, `GeneralPath`.

This class implements the `Shape` interface and represents a geometric path constructed from lines, and quadratic and cubic curves. The three constructors in this class can create the `GeneralPath` object with the default winding rule (`WIND_NON_ZERO`), the given winding rule (`WIND_NON_ZERO` or `WIND_EVEN_ODD`), or the specified initial coordinate capacity. The winding rule specifies how the interior of a path is determined.

To create an empty `GeneralPath` instance call `new GeneralPath ()` and then add segments to the shape by using the following methods:

- `moveTo(float x, float y)` – Moves the current point of the path to the given point
- `lineTo(float x, float y)` – Adds a line segment to the current path
- `quadTo(float ctrlx, float ctrly, float x2, float y2)` – Adds a quadratic curve segment to the current path
- `curveTo(float ctrlx1, float ctrly1, float ctrlx2, float ctrly2, float x3, float y3)` – Adds a cubic curve segment to the current path
- `closePath()` – Closes the current path

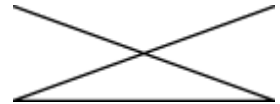
The following example illustrates how to draw a polyline by using `GeneralPath`:

```
// draw GeneralPath (polyline)
int x2Points[] = {0, 100, 0, 100};
int y2Points[] = {0, 50, 50, 0};
GeneralPath polyline =
    new GeneralPath(GeneralPath.WIND_EVEN_ODD, x2Points.length);

polyline.moveTo (x2Points[0], y2Points[0]);

for (int index = 1; index < x2Points.length; index++) {
    polyline.lineTo(x2Points[index], y2Points[index]);
};

graphics2D.draw(polyline);
```



This example illustrates how to draw a polygon by using GeneralPath:

```
// draw GeneralPath (polygon)
int x1Points[] = {0, 100, 0, 100};
int y1Points[] = {0, 50, 50, 0};
GeneralPath polygon =
    new GeneralPath(GeneralPath.WIND_EVEN_ODD, x1Points.length);
polygon.moveTo(x1Points[0], y1Points[0]);

for (int index = 1; index < x1Points.length; index++) {
    polygon.lineTo(x1Points[index], y1Points[index]);
};

polygon.closePath();
graphics2D.draw(polygon);
```



Note that the only difference between two last code examples is the `closePath()` method. This method makes a polygon from a polyline by drawing a straight line back to the coordinates of the last `moveTo`.

3.3 Stroking and Filling Graphics Primitives

- Filling – is a process of painting the shape's interior with solid color or a color gradient, or a texture pattern
- Stroking – is a process of drawing a shape's outline applying stroke width, line style, and color attribute

To apply fancy line styles and fill patterns to geometric primitives change the stroke and paint attributes in the Graphics2D context before rendering. For example, draw a

dashed line by creating an appropriate pen object. To add this pen to the Graphics2D context before you render the line call the setDefaultPen method. Similarly, you apply a gradient fill to a Shape object by creating a Brush object and adding it to the Graphics2D context.

3.4 Examples

The following examples are modified from the examples provided by Tinyline (www.tinyline.com) using Guidebee Graphics 2D API instead of the Tiny2D API.

3.4.1 Alias

The Alias example shows how to antialiasing attributes affects the quality of the drawing. Antialiasing reduces unwanted jagged borders between colors, but it's computationally expensive, costs CPU.

```
//----- PACKAGE -----
package biz.guidebee.example.tiny2d;

//----- IMPORTS -----
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

import biz.guidebee.drawing.*;
import biz.guidebee.geometry.*;

//[----- MAIN CLASS -----]
//----- REVISIONS -----
// Date      Name      Tracking #      Description
// -----
// 01SEP2007 James Shen      Initial Creation
//-----
/**
 * The Alias example shows how to antialiasing attributes affects the quality of
 * the drawing. Antialiasing reduces unwanted jagged borders between colors, but
 * it's computationally expensive, costs CPU.
 */
public class Alias extends MIDlet
{
    /**
     * The display object associated with the MIDlet.
     */
    private Display display;

    /**
     * The Canvas object.
     */
    private Canvas2D canvas;

    /**
     * The graphics object.
     */
    private Graphics2D graphics2D;

    /**
     * The brush object used to fill the ovals.
     */
    private SolidBrush brush=new SolidBrush(Color.GREEN);
}
```

```

public Alias() {

    display = Display.getDisplay(this);
    canvas = new Canvas2D(display);
    graphics2D = canvas.getGraphics();
}

public void startApp() throws MIDletStateChangeException {
    display.setCurrent(canvas);
    //Clear the canvas with white color.
    graphics2D.clear(Color.WHITE);
    AffineTransform matrix=new AffineTransform();
    graphics2D.setAffineTransform(matrix);
    graphics2D.setAntiAliasing(false);
    graphics2D.fillOval(brush,20,40,100,50);

    matrix=new AffineTransform();
    matrix.translate(0,60);
    graphics2D.setAffineTransform(matrix);

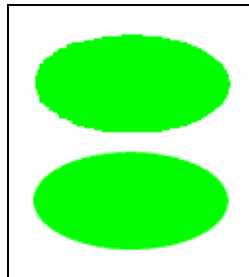
    graphics2D.setAntiAliasing(true);
    graphics2D.fillOval(brush,20,40,100,50);

    graphics2D.invalidate();
}

public void pauseApp() {
}

public void destroyApp(boolean unconditional)
throws MIDletStateChangeException {
    canvas.flush();
    canvas = null;
    System.gc();
}
}

```



3.4.2 Colors

The following example shows how to draw with a single ARGB color.

```

//----- PACKAGE -----
package biz.guidebee.example.tiny2d;

//----- IMPORTS -----
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

```

```

import biz.guidebee.drawing.*;
import biz.guidebee.geometry.*;

//[----- MAIN CLASS -----]
//----- REVISIONS -----
// Date      Name      Tracking #      Description
// -----
// 01SEP2007  James Shen      Initial Creation
//-----
/**
 * The example shows how to use draw with colors.
 */
public class Colors extends MIDlet
{
    /**
     * The display object associated with the MIDlet.
     */
    private Display display;

    /**
     * The Canvas object.
     */
    private Canvas2D canvas;

    /**
     * The graphics object.
     */
    private Graphics2D graphics2D;

    /* Colors */
    /**
     * The solid (full opaque) red color in the ARGB space
     */
    Color redColor    = new Color(0xffff0000);

    /**
     * The semi-opaque green color in the ARGB space (alpha is 0x78)
     */
    Color greenColor = new Color(0x7800ff00,true);

    /**
     * The semi-opaque blue color in the ARGB space (alpha is 0x78)
     */
    Color blueColor = new Color(0x780000ff,true);
    /**
     * The semi-opaque yellow color in the ARGB space ( alpha is 0x78)
     */
    Color yellowColor = new Color(0x78ffff00,true);

    /**
     * The dash array
     */
    int dashArray[] = { 20 ,8 };

    public Colors() {

        display = Display.getDisplay(this);
        canvas = new Canvas2D(display);
        graphics2D = canvas.getGraphics();
    }
}

```

```
public void startApp() throws MIDletStateChangeException {
    display.setCurrent(canvas);
    //Clear the canvas with black color.
    graphics2D.clear(Color.BLACK);

    AffineTransform matrix=new AffineTransform();
    graphics2D.setAffineTransform(matrix);

    SolidBrush brush=new SolidBrush(redColor);
    graphics2D.fillOval(brush,30,60,80,80);

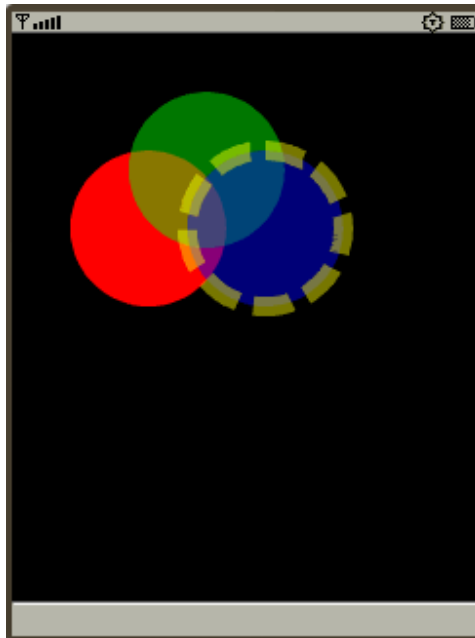
    brush=new SolidBrush(greenColor);
    graphics2D.fillOval(brush,60,30,80,80);

    Pen pen=new Pen(yellowColor,10,Pen.CAP_BUTT,Pen.JOIN_MITER,4,dashArray,0);

    brush=new SolidBrush(blueColor);
    graphics2D.setPenAndBrush(pen,brush);
    graphics2D.fillOval(null,90,60,80,80);
    graphics2D.invalidate();
}

public void pauseApp() {
}

public void destroyApp(boolean unconditional)
throws MIDletStateChangeException {
    canvas.flush();
    canvas = null;
    System.gc();
}
}
```



3.4.3 Beziers

Bezier animates randomly generated path object.

```
//----- PACKAGE -----
package biz.guidebee.example.tiny2d;

//----- IMPORTS -----
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

import biz.guidebee.drawing.*;
import biz.guidebee.geometry.*;

//[----- MAIN CLASS -----]
//----- REVISIONS -----
// Date      Name      Tracking #      Description
// -----
// 01SEP2007  James Shen      Initial Creation
//-----
/**
 * Bazier animates randomly generated path object.
 */
public class Beziers extends MIDlet implements Runnable
{
    /**
     * The display object associated with the MIDlet.
     */
    private Display display;

    /**
     * The Canvas object.
     */
    private Canvas2D canvas;

    /**
     * The graphics object.
     */
    private Graphics2D graphics2D;

    /**
     * The animation thread.
     */
    private Thread thread;
    boolean drawn;

    /**
     * The dirty area.
     */
    Rectangle dirtyRect;

    /**
     * The random number generator.
     */
    static java.util.Random random = new java.util.Random();

    int imgWidth = 100;
    int imgHeight = 100;

    /**
     * The animated path
     */
    GeneralPath path=new GeneralPath();
    AffineTransform mat = new AffineTransform();

    /**
```

```

    * Red brush used to fill the path.
    */
    SolidBrush brush=new SolidBrush(Color.RED);

    private static final int NUMPTS = 6;
    private int animpts[] = new int[NUMPTS * 2];
    private int deltas[] = new int[NUMPTS * 2];
    long startt,endt;

    public Beziers() {
        display = Display.getDisplay(this);
        canvas = new Canvas2D(display);
        graphics2D = canvas.getGraphics();
        path.moveTo(100,100);
        mat.translate(30,30);
        dirtyRect=new Rectangle();
        //Resets the animation data.
        reset(imgWidth,imgHeight);
    }

    public void startApp() throws MIDletStateChangeException {
        display.setCurrent(canvas);
        if (thread == null)
        {
            thread = new Thread(this);
            thread.start();
        }
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional)
        throws MIDletStateChangeException {
        canvas.flush();
        canvas = null;
        System.gc();
    }

    public void start(){
        graphics2D.clear(Color.BLACK);
        if (thread == null)
        {
            thread = new Thread(this);
            thread.start();
        }
    }

    public void stop(){
        thread = null;
    }

    /**
     * Generates new points for the path.
     */
    public void animate(int[] pts, int[] deltas, int i, int limit){
        int newpt = pts[i] + deltas[i];
        if (newpt <= 0) {
            newpt = -newpt;
            deltas[i] = (random.nextInt()&0x00000003) + 2;
        } else if (newpt >= limit) {
            newpt = 2 * limit - newpt;
            deltas[i] = -((random.nextInt()&0x00000003) + 2);
        }
    }

```

```

    }
    pts[i] = newpt;
}

/**
 * Resets the animation data.
 */
public void reset(int w, int h) {
    for (int i = 0; i < animpts.length; i += 2)
    {
        animpts[i + 0] = (random.nextInt() & 0x00000003) * w / 2;
        animpts[i + 1] = (random.nextInt() & 0x00000003) * h / 2;
        deltas[i + 0] = (random.nextInt() & 0x00000003) * 6 + 4;
        deltas[i + 1] = (random.nextInt() & 0x00000003) * 6 + 4;
        if (animpts[i + 0] > w / 2)
        {
            deltas[i + 0] = -deltas[i + 0];
        }
        if (animpts[i + 1] > h / 2)
        {
            deltas[i + 1] = -deltas[i + 1];
        }
    }
}

/**
 * Sets the points of the path and draws and fills the path.
 */
public void drawDemo(int w, int h) {
    Rectangle bounds;
    for (int i = 0; i < animpts.length; i += 2)
    {
        animate(animpts, deltas, i + 0, w);
        animate(animpts, deltas, i + 1, h);
    }
    //Clears the dirty area.
    dirtyRect.setEmpty();
    bounds = path.getBounds();

    Rectangle r = graphics2D.getBounds(mat, 0, bounds);
    //graphics2D.clear(Color.WHITE);
    //Adds the old bounds of the shape to the dirty area.
    dirtyRect = dirtyRect.union(r);

    //Generates the new path data.
    path.reset();
    int[] ctrlpts = animpts;
    int len = ctrlpts.length;
    int prevx = ctrlpts[len - 2];
    int prevy = ctrlpts[len - 1];
    int curx = ctrlpts[0];
    int cury = ctrlpts[1];
    int midx = (curx + prevx) / 2;
    int midy = (cury + prevy) / 2;
    path.moveTo(midx, midy);
    for (int i = 2; i <= ctrlpts.length; i += 2)
    {
        int x1 = (curx + midx) / 2;
        int y1 = (cury + midy) / 2;
        prevx = curx;
        prevy = cury;
        if (i < ctrlpts.length) {
            curx = ctrlpts[i + 0];

```

```

        cury = ctrlpts[i + 1];
    } else {
        curx = ctrlpts[0];
        cury = ctrlpts[1];
    }
    midx = (curx + prevx) / 2;
    midy = (cury + prevy) / 2;
    int x2 = (prevx + midx) / 2;
    int y2 = (prevy + midy) / 2;
    path.curveTo(x1,y1,x2,y2,midx,midy);
}
path.closePath();
bounds = path.getBounds();

// Adds the new bounds of the shape to the dirty area./
Rectangle newBounds=graphics2D.getBounds(mat, 0, bounds);
dirtyRect=dirtyRect.union(newBounds);

// Sets the dirty area.
graphics2D.setClip(dirtyRect);

// clear the clipRect area before production
graphics2D.clearRect(Color.WHITE, graphics2D.getClip());
graphics2D.fill(brush,path);
graphics2D.invalidate();

}

public void run()
{
    Thread me = Thread.currentThread();
    if(!drawn)
    {
        synchronized (this)
        {
            graphics2D.clear(Color.WHITE);
            graphics2D.fill(brush,path);
            graphics2D.invalidate();
            drawn = true;
        }
    }
    while (thread == me)
    {
        drawDemo(imgWidth, imgHeight);
        try {
            thread.sleep(25);
        } catch (Exception e) { break; }
    }
    thread = null;
}
}

```



3.4.4 Dash

The line dash pattern (dashArray) specifies the pattern of dashes and gaps used to stroke paths.

```
//----- PACKAGE -----
package biz.guidebee.example.tiny2d;

//----- IMPORTS -----
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

import biz.guidebee.drawing.*;
import biz.guidebee.geometry.*;

//[----- MAIN CLASS -----]
//----- REVISIONS -----
// Date      Name      Tracking #      Description
// -----
// 01SEP2007 James Shen      Initial Creation
//-----
/**
 * This demo shows line dashes.
 */
public class Dash extends MIDlet
{
    /**
     * The display object associated with the MIDlet.
     */
    private Display display;

    /**
     * The Canvas object.
     */
    private Canvas2D canvas;

    /**
     * The graphics object.
     */
    private Graphics2D graphics2D;

    Color blackColor = new Color(0xff000000);
    int dashArray1[] = { 2, 2 };
    int dashArray2[] = { 6, 6 };
    int dashArray3[] = { 4, 1, 2, 1,1, 6 };

    public Dash() {

        display = Display.getDisplay(this);
        canvas = new Canvas2D(display);
        graphics2D = canvas.getGraphics();
    }

    public void startApp() throws MIDletStateChangeException {
        display.setCurrent(canvas);
        //Clear the canvas with white color.
        graphics2D.clear(Color.WHITE);

        Pen pen=new Pen(blackColor,20,Pen.CAP_BUTT,Pen.JOIN_MITER,4,dashArray1,0);
    }
}
```

```
graphics2D.drawLine(pen,40,60,140,60);

pen=new Pen(blackColor,20,Pen.CAP_BUTT,Pen.JOIN_MITER,4,dashArray2,0);
graphics2D.drawLine(pen,40,100,140,100);

pen=new Pen(blackColor,20,Pen.CAP_BUTT,Pen.JOIN_MITER,4,dashArray3,0);
graphics2D.drawLine(pen,40,140,140,140);

graphics2D.invalidate();
}

public void pauseApp() {
}

public void destroyApp(boolean unconditional)
throws MIDletStateChangeException {
    canvas.flush();
    canvas = null;
    System.gc();
}
}
```



3.4.5 FillRule

The FillRule draws paths with different fillRule values.

```
//----- PACKAGE -----
package biz.guidebee.example.tiny2d;

//----- IMPORTS -----
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

import biz.guidebee.drawing.*;
import biz.guidebee.geometry.*;

//[----- MAIN CLASS -----]
//----- REVISIONS -----
// Date      Name      Tracking #      Description
// -----
// 01SEP2007  James Shen      Initial Creation
//-----
/**
 * The FillRule draws paths with different fillRule values.
 */
public class FillRule extends MIDlet
{
    /**
     * The display object associated with the MIDlet.
     */
    private Display display;

    /**
     * The Canvas object.
     */
    private Canvas2D canvas;

    /**
     * The graphics object.
     */
    private Graphics2D graphics2D;

    AffineTransform mat1, mat2;

    /* The path. */
    GeneralPath path;

    /** Colors */
    Color redColor = new Color(0xffff0000);

    /* The path data. */
    static char[] pathdata =
        "M 110 75 l 50 160 l -130 -100 l 160 0 l -130 100 z".toCharArray();

    public FillRule() {

        display = Display.getDisplay(this);
        canvas = new Canvas2D(display);
        graphics2D = canvas.getGraphics();
        path=graphics2D.toPath(new String(pathdata));
    }
}
```

```

    mat1=new AffineTransform();
    mat1.scale(0.5,0.5);
    mat2=new AffineTransform();

    mat2.translate(80,0);
    mat2.scale(0.5,0.5);
}

public void startApp() throws MIDletStateChangeException {
    display.setCurrent(canvas);
    //Clear the canvas with white color.
    graphics2D.clear(Color.WHITE);

    SolidBrush brush=new SolidBrush(redColor);
    brush.setWindingRule(Brush.WIND_EVEN_ODD);
    graphics2D.setAffineTransform(mat1);
    graphics2D.fill(brush,path);

    graphics2D.setAffineTransform(mat2);
    brush.setWindingRule(Brush.WIND_NON_ZERO);
    graphics2D.fill(brush,path);
    graphics2D.invalidate();
}

public void pauseApp() {
}

public void destroyApp(boolean unconditional)
    throws MIDletStateChangeException {
    canvas.flush();
    canvas = null;
    System.gc();
}
}

```



3.4.6 LineCap

The LineCap shows how to draw lines with different line cap styles.

```

//----- PACKAGE -----
package biz.guidebee.example.tiny2d;

//----- IMPORTS -----
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

import biz.guidebee.drawing.*;
import biz.guidebee.geometry.*;

//[----- MAIN CLASS -----]
////////////////////////////////////

```

```
//----- REVISIONS -----
// Date      Name      Tracking #      Description
// -----
// 01SEP2007 James Shen      Initial Creation
////////////////////////////////////
/**
 * The LineCap shows how to draw lines with different line cap styles.
 */
public class LineCap extends MIDlet
{
    /**
     * The display object associated with the MIDlet.
     */
    private Display display;

    /**
     * The Canvas object.
     */
    private Canvas2D canvas;

    /**
     * The graphics object.
     */
    private Graphics2D graphics2D;

    Color blackColor = new Color(0xff000000);
    Color whiteColor = new Color(0xffffffff);

    public LineCap() {

        display = Display.getDisplay(this);
        canvas = new Canvas2D(display);
        graphics2D = canvas.getGraphics();
    }

    public void startApp() throws MIDletStateChangeException {
        display.setCurrent(canvas);
        //Clear the canvas with white color.
        graphics2D.clear(Color.WHITE);

        Pen pen=new Pen(blackColor,20,Pen.CAP_BUTT,Pen.JOIN_MITER);
        graphics2D.drawLine(pen,40,60,140,60);
        pen=new Pen(whiteColor,1);
        graphics2D.drawLine(pen,40,60,140,60);

        pen=new Pen(blackColor,20,Pen.CAP_ROUND,Pen.JOIN_MITER);
        graphics2D.drawLine(pen,40,100,140,100);
        pen=new Pen(whiteColor,1);
        graphics2D.drawLine(pen,40,100,140,100);

        pen=new Pen(blackColor,20,Pen.CAP_SQUARE,Pen.JOIN_MITER);
        graphics2D.drawLine(pen,40,140,140,140);
        pen=new Pen(whiteColor,1);
        graphics2D.drawLine(pen,40,140,140,140);

        graphics2D.invalidate();
    }

    public void pauseApp() {
    }
}
```

```

public void destroyApp(boolean unconditional)
    throws MIDletStateChangeException {
    canvas.flush();
    canvas = null;
    System.gc();
}
    
```



3.4.7 LineJoin

The LineJoin shows how to specify line join styles.

```

//----- PACKAGE -----
package biz.guidebee.example.tiny2d;

//----- IMPORTS -----
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

import biz.guidebee.drawing.*;
import biz.guidebee.geometry.*;

//[----- MAIN CLASS -----]
//----- REVISIONS -----
// Date          Name          Tracking #          Description
// -----
// 01SEP2007    James Shen          Initial Creation
//-----
/**
 * The LineJoin shows how to specify line join styles.
 */
public class LineJoin extends MIDlet
{
    /**
     * The display object associated with the MIDlet.
     */
    private Display display;

    /**
     * The Canvas object.
     */
    private Canvas2D canvas;

    /**
     * The graphics object.
     */
    private Graphics2D graphics2D;
}
    
```

```
Color blackColor = new Color(0xff000000);
Color whiteColor = new Color(0xffffffff);

GeneralPath path=new GeneralPath();

public LineJoin() {

    display = Display.getDisplay(this);
    canvas = new Canvas2D(display);
    graphics2D = canvas.getGraphics();

    path.moveTo(40, 60);
    path.lineTo(90, 20);
    path.lineTo(140, 60);
}

public void startApp() throws MIDletStateChangeException {
    display.setCurrent(canvas);
    //Clear the canvas with white color.
    graphics2D.clear(Color.WHITE);
    AffineTransform matrix=new AffineTransform();
    graphics2D.setAffineTransform(matrix);
    Pen pen=new Pen(blackColor,20,Pen.CAP_BUTT,Pen.JOIN_MITER);
    graphics2D.draw(pen,path);
    pen=new Pen(whiteColor,1);
    graphics2D.draw(pen,path);

    matrix.translate(0,50);
    graphics2D.setAffineTransform(matrix);

    pen=new Pen(blackColor,20,Pen.CAP_BUTT,Pen.JOIN_ROUND);
    graphics2D.draw(pen,path);
    pen=new Pen(whiteColor,1);
    graphics2D.draw(pen,path);

    matrix=new AffineTransform();
    matrix.translate(0,100);
    graphics2D.setAffineTransform(matrix);

    pen=new Pen(blackColor,20,Pen.CAP_BUTT,Pen.JOIN_BEVEL);
    graphics2D.draw(pen,path);
    pen=new Pen(whiteColor,1);
    graphics2D.draw(pen,path);
    graphics2D.invalidate();
}

public void pauseApp() {
}

public void destroyApp(boolean unconditional)
    throws MIDletStateChangeException {
    canvas.flush();
    canvas = null;
    System.gc();
}
}
```



3.4.8 Lines

The Lines shows how to draw lines.

```

//----- PACKAGE -----
package biz.guidebee.example.tiny2d;

//----- IMPORTS -----
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

import biz.guidebee.drawing.*;
import biz.guidebee.geometry.*;

//[----- MAIN CLASS -----]
///////////////////////////////////////////////////////////////////
//----- REVISIONS -----
// Date      Name      Tracking #      Description
// -----
// 01SEP2007 James Shen      Initial Creation
///////////////////////////////////////////////////////////////////
/**
 * The Lines shows how to draw lines.
 */
public class Lines extends MIDlet
{
    /**
     * The display object associated with the MIDlet.
     */
    private Display display;

    /**
     * The Canvas object.
     */
    private Canvas2D canvas;

    /**
     * The graphics object.
     */
    private Graphics2D graphics2D;

    Color greenColor = new Color(0xff00ff00);

    public Lines() {

        display = Display.getDisplay(this);
    }
}

```

```
        canvas = new Canvas2D(display);
        graphics2D = canvas.getGraphics();
    }

    public void startApp() throws MIDletStateChangeException {
        display.setCurrent(canvas);
        //Clear the canvas with white color.
        graphics2D.clear(Color.WHITE);

        Pen pen=new Pen(greenColor,1);
        graphics2D.drawLine(pen,20,150,60,50);

        pen=new Pen(greenColor,2);
        graphics2D.drawLine(pen,40,150,80,50);

        pen=new Pen(greenColor,3);
        graphics2D.drawLine(pen,60,150,100,50);

        pen=new Pen(greenColor,5);
        graphics2D.drawLine(pen,80,150,120,50);

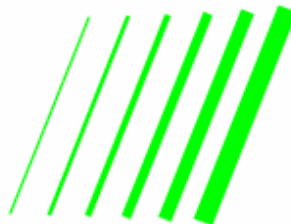
        pen=new Pen(greenColor,7);
        graphics2D.drawLine(pen,100,150,140,50);

        pen=new Pen(greenColor,10);
        graphics2D.drawLine(pen,120,150,160,50);

        graphics2D.invalidate();
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional)
        throws MIDletStateChangeException {
        canvas.flush();
        canvas = null;
        System.gc();
    }
}
```



3.4.9 Ovals

The Ovals shows how to draw ovals.

```

//----- PACKAGE -----
package biz.guidebee.example.tiny2d;

//----- IMPORTS -----
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

import biz.guidebee.drawing.*;
import biz.guidebee.geometry.*;

//[----- MAIN CLASS -----]
//----- REVISIONS -----
// Date          Name          Tracking #          Description
// -----
// 01SEP2007    James Shen          Initial Creation
//-----
/**
 * The Ovals shows how to draw ovals.
 */
public class Ovals extends MIDlet
{
    /**
     * The display object associated with the MIDlet.
     */
    private Display display;

    /**
     * The Canvas object.
     */
    private Canvas2D canvas;

    /**
     * The graphics object.
     */
    private Graphics2D graphics2D;

    AffineTransform mat1;

    /** Colors */
    Color redColor    = new Color(0x96ff0000,true);
    Color greenColor  = new Color(0xff00ff00);

    //static char xformdata1[] = "translate(30 40) rotate(-30)".toCharArray();

    public Ovals() {

        display = Display.getDisplay(this);
        canvas = new Canvas2D(display);
        graphics2D = canvas.getGraphics();
        mat1=new AffineTransform();
        mat1.translate(30,40);
        mat1.rotate(-30 * Math.PI/180.0);

    }
}

```

```

public void startApp() throws MIDletStateChangeException {
    display.setCurrent(canvas);
    //Clear the canvas with white color.
    graphics2D.clear(Color.WHITE);
    graphics2D.setAffineTransform(new AffineTransform());
    SolidBrush brush=new SolidBrush(greenColor);
    brush.setWindingRule(Brush.WIND_EVEN_ODD);
    graphics2D.fillOval(brush,20,60,100,50);

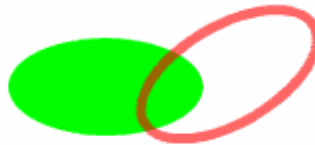
    Pen pen=new Pen(redColor,5);
    graphics2D.setAffineTransform(mat1);
    graphics2D.drawOval(pen,20,60,100,50);

    graphics2D.invalidate();
}

public void pauseApp() {
}

public void destroyApp(boolean unconditional)
throws MIDletStateChangeException {
    canvas.flush();
    canvas = null;
    System.gc();
}
}

```



3.4.10 Paths

The Paths shows how to draw paths.

```

//----- PACKAGE -----
package biz.guidebee.example.tiny2d;

//----- IMPORTS -----
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

import biz.guidebee.drawing.*;
import biz.guidebee.geometry.*;

//[----- MAIN CLASS -----]
//----- REVISIONS -----
// Date      Name      Tracking #      Description
// -----
// 01SEP2007 James Shen      Initial Creation
//-----
/**
 * The Paths shows how to draw paths.
 */

```

```
public class Paths extends MIDlet
{
    /**
     * The display object associated with the MIDlet.
     */
    private Display display;

    /**
     * The Canvas object.
     */
    private Canvas2D canvas;

    /**
     * The graphics object.
     */
    private Graphics2D graphics2D;

    AffineTransform mat1;

    /* The path. */
    GeneralPath path;

    /** Colors */
    Color redColor    = new Color(0x96ff0000,true);
    Color greenColor  = new Color(0xff00ff00);
    Color blueColor   = new Color(0x750000ff,true);

    static char[] pathdata =
        "M 60 20 Q -40 70 60 120 Q 160 70 60 20 z".toCharArray();

    public Paths() {
        display = Display.getDisplay(this);
        canvas = new Canvas2D(display);
        graphics2D = canvas.getGraphics();
        mat1=new AffineTransform();
        mat1.translate(30,40);
        mat1.rotate(-30 * Math.PI/180.0);
        path=Graphics2D.toPath(new String(pathdata));
    }

    public void startApp() throws MIDletStateChangeException {
        display.setCurrent(canvas);
        //Clear the canvas with white color.
        graphics2D.clear(Color.WHITE);
        graphics2D.setAffineTransform(new AffineTransform());
        SolidBrush brush=new SolidBrush(greenColor);
        graphics2D.fill(brush,path);
        graphics2D.setAffineTransform(mat1);

        brush=new SolidBrush(blueColor);
        Pen pen=new Pen(redColor,5);
        graphics2D.setPenAndBrush(pen,brush);
        graphics2D.draw(null,path);
        graphics2D.invalidate();
    }

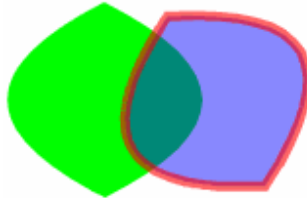
    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional)
        throws MIDletStateChangeException {
        canvas.flush();
        canvas = null;
    }
}
```

```

        System.gc();
    }
}

```



3.4.11 Polys

The Polys shows how to draw polygons and polylines.

```

//----- PACKAGE -----
package biz.guidebee.example.tiny2d;

//----- IMPORTS -----
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

import biz.guidebee.drawing.*;
import biz.guidebee.geometry.*;

//[----- MAIN CLASS -----]
////////////////////////////////////////////////////////////////////
//----- REVISIONS -----
// Date      Name           Tracking #      Description
// -----
// 01SEP2007 James Shen           Initial Creation
//////////////////////////////////////////////////////////////////
/**
 * The Polys shows how to draw polygons and polylines
 */
public class Polys extends MIDlet
{
    /**
     * The display object associated with the MIDlet.
     */
    private Display display;

    /**
     * The Canvas object.
     */
    private Canvas2D canvas;

    /**
     * The graphics object.
     */
    private Graphics2D graphics2D;

    AffineTransform mat1;

    /** Colors */
    Color redColor = new Color(0x96ff0000,true);
    Color greenColor = new Color(0xff00ff00);

```

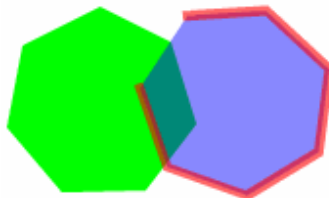
```
Color blueColor = new Color(0x750000ff,true);
Polyline polyline;
Polygon polygon;
static char pointsdata1[] =
"59,45,95,63,108,105,82,139,39,140,11,107,19,65".toCharArray();

public Polys() {
    display = Display.getDisplay(this);
    canvas = new Canvas2D(display);
    graphics2D = canvas.getGraphics();
    mat1=new AffineTransform();
    mat1.translate(30,40);
    mat1.rotate(-30 * Math.PI/180.0);
    polyline=new Polyline();
    polygon=new Polygon();
    Point[] points=Graphics2D.toPoints(new String(pointsdata1));
    for(int i=0;i<points.length;i++){
        polyline.addPoint(points[i].x,points[i].y);
        polygon.addPoint(points[i].x,points[i].y);
    }
}

public void startApp() throws MIDletStateChangeException {
    display.setCurrent(canvas);
    //Clear the canvas with white color.
    graphics2D.clear(Color.WHITE);
    graphics2D.setAffineTransform(new AffineTransform());
    SolidBrush brush=new SolidBrush(greenColor);
    graphics2D.fillPolygon(brush,polygon);
    graphics2D.setAffineTransform(mat1);
    brush=new SolidBrush(blueColor);
    Pen pen=new Pen(redColor,5);
    graphics2D.setPenAndBrush(pen,brush);
    graphics2D.drawPolyline(null,polyline);
    graphics2D.invalidate();
}

public void pauseApp() {
}

public void destroyApp(boolean unconditional)
throws MIDletStateChangeException {
    canvas.flush();
    canvas = null;
    System.gc();
}
}
```



3.4.12 Gradients & Pattern Brush

Shapes or text can be filled with gradient colors or patterns (images). Beside SolidBrush, there are other three brushes:

- LinearGradientBrush fill a Shape with a linear color gradient pattern.
- RadialGradientBrush fill a shape with a circular radial color gradient pattern.
- TextureBrush fill a Shape with a texture that is specified as an Image.

The following example shows how to use LinearGradientBrush and RadialGradientBrush.

```

package biz.guidebee.example.tiny2d;

//----- IMPORTS -----
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

import biz.guidebee.drawing.*;
import biz.guidebee.geometry.*;
//[----- MAIN CLASS -----]
//----- REVISIONS -----
// Date      Name      Tracking #      Description
// -----
// 01SEP2007  James Shen      Initial Creation
//-----
/**
 * This examples show how to fill with gradient brushes.
 */
public class Gradients extends MIDlet
{
    /**
     * The display object associated with the MIDlet.
     */
    private Display display;

    /**
     * The Canvas object.
     */
    private Canvas2D canvas;

    /**
     * The graphics object.
     */
    private Graphics2D graphics2D;

    /* The linear gradient color */
    LinearGradientBrush brush1;
    /* The radial gradient color */
    RadialGradientBrush brush2;
    /* The second radial gradient color */
    RadialGradientBrush brush3;

    char [] engText = "Brush".toCharArray();
    FontEx font=FontEx.getSystemFont();
    int fontSize =44;
    int X =20;
    int Y =50;

    public Gradients() {

```

```
display = Display.getDisplay(this);
canvas = new Canvas2D(display);
graphics2D = canvas.getGraphics();
int []fractions=new int[]{13,242};
Color []colors=new Color[]{new Color(0xffff6600),new Color(0xfffff66)};
brush1=new LinearGradientBrush(50,50,150,125,fractions,colors,Brush.REFLECT);
fractions=new int[]{13,128,255};
colors=new Color[]{new Color(0xffff6600),new Color(0xfffff66),
    new Color(0xffff6600)};
brush2=new RadialGradientBrush(90,100,50,fractions,colors);
fractions=new int[]{0,255};
colors=new Color[]{new Color(0xFFFFF00),new Color(0xFF000000)};
brush3=new RadialGradientBrush(50,50,100,fractions,colors);
}

public void startApp() throws MIDletStateChangeException {
display.setCurrent(canvas);
//Clear the canvas with white color.
graphics2D.clear(Color.BLACK);
graphics2D.fillRect(brush1,new Rectangle(10,75,120,80));
Pen pen=new Pen(brush2,8);
graphics2D.drawOval(pen,20,60,100,50);
graphics2D.setDefaultBrush(brush3);
graphics2D.drawChars(font,fontSize,engText,0,engText.length,X,Y,
    FontEx.TEXT_ANCHOR_START);
graphics2D.invalidate();
}

public void pauseApp() {
}

public void destroyApp(boolean unconditional)
throws MIDletStateChangeException {
canvas.flush();
canvas = null;
System.gc();
}
}
```



The following example shows how to fill with TextureBrush.

```

//----- PACKAGE -----
package biz.guidebee.example.tiny2d;

//----- IMPORTS -----
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

import biz.guidebee.drawing.*;
import biz.guidebee.geometry.*;
//[----- MAIN CLASS -----]
//----- REVISIONS -----
// Date      Name      Tracking #      Description
// -----
// 01SEP2007 James Shen      Initial Creation
//-----
/**
 * This examples show how to fill with patterns.
 */
public class Patterns extends MIDlet
{
    /**
     * The display object associated with the MIDlet.
     */
    private Display display;

    /**
     * The Canvas object.
     */
    private Canvas2D canvas;

    /**
     * The graphics object.
     */
    private Graphics2D graphics2D;

    TextureBrush brush1;
    TextureBrush brush2;
    TextureBrush brush3;

    AffineTransform matrix1=new AffineTransform();
    AffineTransform matrix2=new AffineTransform();

    public Patterns() {

        display = Display.getDisplay(this);
        canvas = new Canvas2D(display);
        graphics2D = canvas.getGraphics();
        try{

            brush1=new TextureBrush(Image.createImage(this.getClass().
                getResourceAsStream("/brick.png")));
            brush2=new TextureBrush(Image.createImage(this.getClass().
                getResourceAsStream("/bird.png")));
            brush3=new TextureBrush(Image.createImage(this.getClass().
                getResourceAsStream("/bird.png"),127);
        }
        catch(Exception e){
            System.out.println(e.getMessage());
        }
    }
}

```

```
matrix2.translate(50,50);  
  
}  
  
public void startApp() throws MIDletStateChangeException {  
    display.setCurrent(canvas);  
    //Clear the canvas with white color.  
    graphics2D.clear(Color.BLACK);  
    graphics2D.setAffineTransform(matrix1);  
    graphics2D.fillRect(brush1,new Rectangle(20,50,100,100));  
    graphics2D.fillOval(brush2,10,10,80,80);  
    graphics2D.setAffineTransform(matrix2);  
    graphics2D.fillOval(brush3,10,10,80,80);  
    graphics2D.invalidate();  
}  
  
public void pauseApp() {  
}  
  
public void destroyApp(boolean unconditional)  
    throws MIDletStateChangeException {  
    canvas.flush();  
    canvas = null;  
    System.gc();  
}  
}
```



4.0 Working with Text and Font

This section describes how to draw text, how to set Font.

4.1 Selecting a Font

Guidebee Graphics 2D API support two kinds of Font, one is the Java ME Font (javax.microedition.lcdiui.Font), the other one is provided by the Graphics 2D API (biz.guidebee.drawing.FontEx).

The major difference between Font and FontEx is that FontEx supports transformation of the font glyph. It has getGlyphArray and getGlyphOutline which can return font glyph as general shape object. So that any affine transformation applies to Shape can apply to FontEx also.

FontEx has constructor which takes java.io.InputStream as the input parameter, which can be used to read the font data from files.

FontEx also provides one default font .The static method getSystemFont returns the default FontEx object which can be used to draw English text to the canvas.

The Font file has the following format:

Header	Stores the font name, font metrics information
Index	Index to the Data area for each character
Data	SVG string for each character

Currently Guidebee Graphics2D API provides the following Font types.



The following examples use JSR75 (FileConnection) API load above font type and display different font type on screen.

Note: The InputStream has to support random access (that is mark/reset need to be supported by the InputStream) for the FontEx class.

```
//----- PACKAGE -----
package biz.guidebee.example.drawing2d;

//----- IMPORTS -----
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.file.*;
import javax.microedition.io.*;
import java.io.*;

import biz.guidebee.drawing.*;
import biz.guidebee.geometry.*;

//[----- MAIN CLASS -----]
//----- REVISIONS -----
// Date      Name      Tracking #      Description
// -----
// 01SEP2007  James Shen      Initial Creation
//-----
/**
 * The example displays different font types.
 */
public class Fonts extends MIDlet
{
    /**
     * The display object associated with the MIDlet.
     */
    private Display display;

    /**
     * The Canvas object.
     */
    private Canvas2D canvas;

    /**
     * The graphics object.
     */
    private Graphics2D graphics2D;

    private static String fontFileName[]={
        "arial.fon",
        "courier.fon",
        "elephant.fon",
        "impact.fon",
        "georgia.fon",
        "rockwell.fon",
        "roman.fon",
        "serif.fon",
        "verdana.fon",
        "youyuan.fon",
        "xinwei.fon",
        "xinsong.fon",
        "xingkai.fon",
        "songti.fon",
        "lishu.fon",
        "fangsong.fon",
        "heiti.fon"
    };
};
```

```

public Fonts() {

    display = Display.getDisplay(this);
    canvas = new Canvas2D(display);
    graphics2D = canvas.getGraphics();
}

public void startApp() throws MIDletStateChangeException {
    display.setCurrent(canvas);
    //Clear the canvas with white color.
    graphics2D.clear(Color.WHITE);
    AffineTransform matrix=new AffineTransform();
    graphics2D.setAffineTransform(matrix);
    FontEx font=FontEx.getSystemFont();
    FileConnection fconn=null;
    char [] longLine = null;
    char [] errorLine = "Error! ".toCharArray();
    graphics2D.setPenAndBrush(new Pen(Color.BLACK,1),
        new SolidBrush(Color.BLACK));
    try {
        String fontName;
        for(int i=0;i<fontFileName.length;i++){
            fontName="file:///fonts/" + fontFileName[i];
            fconn = (FileConnection)Connector.open(fontName);
            font=new FontEx(fconn.openInputStream());
            System.out.println(font.getName());
            longLine=font.getName().toCharArray();
            graphics2D.drawChars(font,16,longLine,0,longLine.length,20,16+16*i);
            fconn.close();
        }
    } catch (IOException ioe) {
        graphics2D.drawChars(font,16,errorLine,0,errorLine.length,10,20);
    }

    graphics2D.invalidate();
}

public void pauseApp() {
}

public void destroyApp(boolean unconditional)
throws MIDletStateChangeException {
    canvas.flush();
    canvas = null;
    System.gc();
}
}

```

Note: Chinese character may not be shown on no-Chinese OS. So exception may be thrown running this MIDlet. To solve this problem, comments out those Chinese font type in the array fontFileName.

4.2 Measuring Text and Text Direction

To measure text, call FontEx's static methods

```

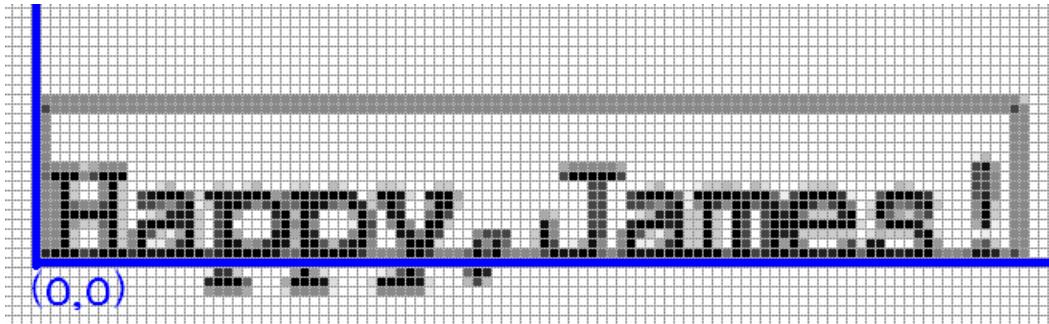
public int charsWidth(int fontSize,
                    char[] ch,
                    int offset,

```

```
int length,
int tdir)
```

Text can be displayed from Left to Right, Right to Left and from Top to bottom.

Method drawChar() specify the start point (x, y) where text start to be drawn, the anchor point of text is left-bottom of the baseline of a font. As shown below, the (0,0) is the anchor point of the string “Happy, James!”.



The following example shows how to measure string and how to display a string in different directions.

```
//----- PACKAGE -----
package biz.guidebee.example.drawing2d;

//----- IMPORTS -----
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

import biz.guidebee.drawing.*;
import biz.guidebee.geometry.*;

//[----- MAIN CLASS -----]
///////////////////////////////////////////////////////////////////
//----- REVISIONS -----
// Date      Name      Tracking #      Description
// -----
// 01SEP2007  James Shen      Initial Creation
///////////////////////////////////////////////////////////////////
/**
 * This demo shows font direction and measure string.
 */
public class Fonts extends MIDlet
{
    /**
     * The display object associated with the MIDlet.
     */
    private Display display;

    /**
     * The Canvas object.
     */
    private Canvas2D canvas;

    /**
     * The graphics object.
     */
    private Graphics2D graphics2D;
```

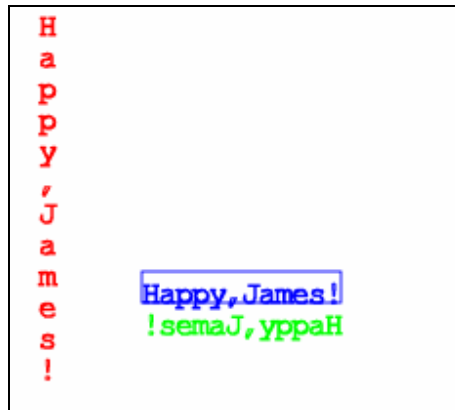
```
public Fonts() {
    display = Display.getDisplay(this);
    canvas = new Canvas2D(display);
    graphics2D = canvas.getGraphics();
}

public void startApp() throws MIDletStateChangeException {
    display.setCurrent(canvas);
    //Clear the canvas with white color.
    graphics2D.clear(Color.WHITE);
    AffineTransform matrix=new AffineTransform();
    graphics2D.setAffineTransform(matrix);
    FontEx font=FontEx.getSystemFont();
    int fontSize=16;
    char [] longLine = "Happy,James!".toCharArray();
    int stringWidth=font.charsWidth(fontSize,longLine,0,longLine.length,
        FontEx.TEXT_DIR_LR);
    int canvasWidth=canvas.getWidth();
    int canvasHeight=canvas.getHeight();
    graphics2D.setDefaultPen(new Pen(Color.BLUE,1));
    Rectangle rectangle=new Rectangle((canvasWidth-stringWidth)/2,
        (canvasHeight-16)/2,stringWidth,16);
    graphics2D.drawRectangle(null,rectangle);
    graphics2D.setTextDirection(FontEx.TEXT_DIR_LR);
    graphics2D.drawChars(font,fontSize,longLine,0,longLine.length,rectangle.x,
        rectangle.y+fontSize);
    graphics2D.setDefaultPen(new Pen(Color.GREEN,1));
    graphics2D.setTextDirection(FontEx.TEXT_DIR_RL);
    graphics2D.drawChars(font,fontSize,longLine,0,longLine.length,
        rectangle.x+stringWidth,
        rectangle.y+fontSize*2);
    graphics2D.setDefaultPen(new Pen(Color.RED,1));
    graphics2D.setTextDirection(FontEx.TEXT_DIR_TB);
    graphics2D.drawChars(font,fontSize,longLine,0,longLine.length,16,
        16);

    graphics2D.invalidate();
}

public void pauseApp() {
}

public void destroyApp(boolean unconditional)
throws MIDletStateChangeException {
    canvas.flush();
    canvas = null;
    System.gc();
}
}
```



4.3 Glyphs

FontEx has sets of shapes that are associated with characters. It has two methods `getGlyphArray` and `getGlyphOutline` which can be used to return `Shape` associated with each character or the whole string.

```
package biz.guidebee.example.drawing2d;

//----- IMPORTS -----
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

import biz.guidebee.drawing.*;
import biz.guidebee.geometry.*;

//[----- MAIN CLASS -----]
//----- REVISIONS -----
// Date      Name      Tracking #      Description
// -----
// 01SEP2007  James Shen      Initial Creation
//-----
/**
 * This demo shows font glyphs
 */
public class Glyphs extends MIDlet
{
    /**
     * The display object associated with the MIDlet.
     */
    private Display display;

    /**
     * The Canvas object.
     */
    private Canvas2D canvas;

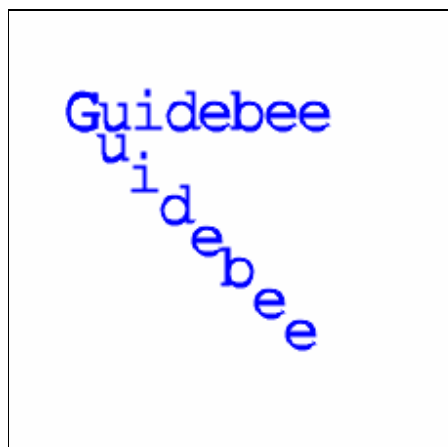
    /**
     * The graphics object.
     */
    private Graphics2D graphics2D;
    public Glyphs() {
```

```
        display = Display.getDisplay(this);
        canvas = new Canvas2D(display);
        graphics2D = canvas.getGraphics();
    }

    public void startApp() throws MIDletStateChangeException {
        display.setCurrent(canvas);
        //Clear the canvas with white color.
        graphics2D.clear(Color.WHITE);
        AffineTransform matrix=new AffineTransform();
        graphics2D.setAffineTransform(matrix);
        FontEx font=FontEx.getSystemFont();
        int fontSize=32;
        int x=32;
        int y=32;
        char [] longLine = "Guidebee".toCharArray();
        Shape shape=font.getGlyphOutline(fontSize,longLine,0,longLine.length,
            x,y,FontEx.TEXT_DIR_LR);
        graphics2D.setPenAndBrush(new Pen(Color.BLUE,1),new SolidBrush(Color.BLUE));
        graphics2D.draw(null,shape);
        Shape []shapes=font.getGlyphArray(fontSize,longLine,0,longLine.length,
            FontEx.TEXT_DIR_LR);
        AffineTransform matrix1=new AffineTransform();
        for(int i=0;i<shapes.length;i++){
            matrix1.setToIdentity();
            matrix1.translate(x+i*16,y+i*16);
            Shape shapel=matrix1.createTransformedShape(shapes[i]);
            graphics2D.draw(null,shapel);
        }
        graphics2D.invalidate();
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional)
        throws MIDletStateChangeException {
        canvas.flush();
        canvas = null;
        System.gc();
    }
}
```



5.0 Working with Images

Images are described by a width and a height, measured in pixels, Guidebee Graphics 2D APIs support two Image classes,

- `javax.microedition.lcdui.Image`
- `biz.guidebee.drawing.Icon`

5.1 Reading/Loading an image or an Icon

`javax.microedition.lcdui.Image` is a classes within package `javax.microedition.lcdui` , a lot of tutorials are available. So I'll not repeat it here.

Icon class represents a Windows icon, which is a small bitmap image used to represent an object.

The following code shows how to load icons from resource file.

```
try{
    plane=new Icon(this.getClass().getResourceAsStream("/PLANE.ICO"));
    hand=new Icon(this.getClass().getResourceAsStream("/POINT04.ICO"));
}catch (Exception e){
    System.out.println(e.getMessage());
}
```

5.2 Drawing Images or Icons

Class `Graphics2D` provides a bunch of overloaded `drawImage` methods:

- `void drawImage(javax.microedition.lcdui.Image image, int dstX, int dstY)`
- `void drawImage(javax.microedition.lcdui.Image image, int dstX, int dstY, int transparency)`
- `void drawImage(javax.microedition.lcdui.Image image, int dstX, int dstY, int transparency, int alpha)`
- `void drawImage(javax.microedition.lcdui.Image image, int dstX, int dstY, int srcX, int srcY,`

- ```
 int srcWidth,
 int srcHeight)
● void drawImage(javax.microedition.lcdui.Image image,
 int dstX,
 int dstY,
 int srcX,
 int srcY,
 int srcWidth,
 int srcHeight,
 Color transparency)
● void drawImage(javax.microedition.lcdui.Image image,
 int dstX,
 int dstY,
 int srcX,
 int srcY,
 int srcWidth,
 int srcHeight,
 int transparency)
```

and one method to draw Icon

- `void drawIcon(Icon icon,int dstX, int dstY)`

image - Image object to be drawn.

Icon - Icon object to be drawn

dstX - x-coordinate of the upper-left corner of the drawn image.

dstY - y-coordinate of the upper-left corner of the drawn image.

srcX - x-coordinate of the upper-left corner of the portion of the source image to draw.

srcY - y-coordinate of the upper-left corner of the portion of the source image to draw.

srcWidth - Width of the portion of the source image to draw.

srcHeight - Height of the portion of the source image to draw.

transparency - specify the transparent color of the image.

alpha - alpha value of the image.

The following code example divides an image into four quadrants and randomly draws each quadrant of the source image into a different quadrant of the destination.

```
//----- PACKAGE -----
package biz.guidebee.example.drawing2d;

//----- IMPORTS -----
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

import biz.guidebee.drawing.*;
import biz.guidebee.geometry.*;
import java.util.Random;

//[----- MAIN CLASS -----]
//----- REVISIONS -----
// Date Name Tracking # Description
// -----
// 01SEP2007 James Shen Initial Creation
//-----
/**
 * this example divides an image into four quadrants and randomly draws each
 * quadrant of the source image into a different quadrant of the destination.
 */
public class JumbledImage extends MIDlet implements CommandListener
{
 /**
 * The display object associated with the MIDlet.
 */
 private Display display;

 /**
 * The Canvas object.
 */
 private Canvas2D canvas;

 /**
 * The graphics object.
 */
 private Graphics2D graphics2D;

 private int numlocs = 2;
 private int numcells = numlocs*numlocs;
 private int[] cells;
 private Image bi;
 int w, h, cw, ch;
 int offX,offY;

 Command cmdRedraw=new Command("Jumble",Command.SCREEN,1);

 public void commandAction(Command c,
 Displayable d){
 if(c==cmdRedraw){
 jumble();
 redraw();
 }
 }
}
```

```

public JumbledImage() {

 display = Display.getDisplay(this);
 canvas = new Canvas2D(display);
 graphics2D = canvas.getGraphics();
 try{
 bi=Image.createImage(this.getClass().
 getResourceAsStream("/duke_skateboard.jpg"));
 w = bi.getWidth();
 h = bi.getHeight();
 }catch (Exception e){
 System.out.println(e.getMessage());
 }

 cw = w/numlocs;
 ch = h/numlocs;
 cells = new int[numcells];
 for (int i=0;i<numcells;i++) {
 cells[i] = i;
 }

 offX=(canvas.getWidth()-w)/2;
 offY=(canvas.getHeight()-h)/2;

 canvas.addCommand(cmdRedraw);
 canvas.setCommandListener(this);
}

void jumble() {
 Random rand = new Random();
 int ri;
 for (int i=0; i<numcells; i++) {
 while ((ri = rand.nextInt(numlocs)) == i);

 int tmp = cells[i];
 cells[i] = cells[ri];
 cells[ri] = tmp;
 }
}

void redraw(){
 graphics2D.clear(Color.WHITE);

 int dx, dy;
 for (int x=0; x<numlocs; x++) {
 int sx = x*cw;
 for (int y=0; y<numlocs; y++) {
 int sy = y*ch;
 int cell = cells[x*numlocs+y];
 dx = (cell / numlocs) * cw;
 dy = (cell % numlocs) * ch;
 graphics2D.drawImage(bi,
 dx+offX, dy+offY,
 sx, sy, cw, ch);
 }
 }

 graphics2D.invalidate();
}

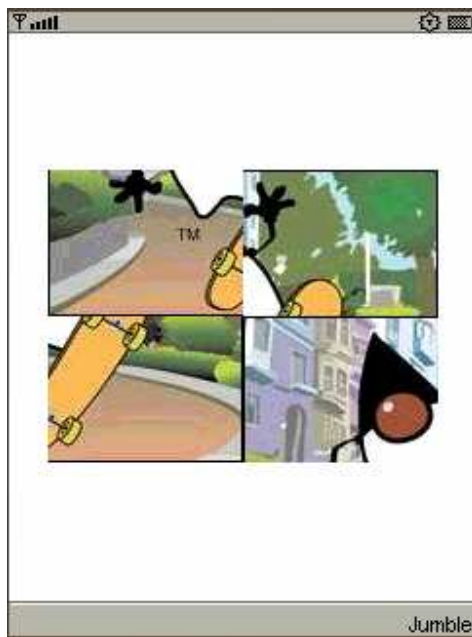
public void startApp() throws MIDletStateChangeException {
 display.setCurrent(canvas);
 //Clear the canvas with white color.
 redraw();
}

```

```
}

public void pauseApp() {
}

public void destroyApp(boolean unconditional)
 throws MIDletStateChangeException {
 canvas.flush();
 canvas = null;
 System.gc();
}
}
```



Icons are actually images with fixed width and height (16X16 or 32X32 etc). Icon classes provides methods to convert Icon to normal Images :toImage and toImages. Standard window Icon contains two images , one is 16X16 small icon, the other is 32X32 icon. toImage convert to default 32X32 icon to image, toImages convert the 2 icon to an images array. [0] is 32X32 icon, [1] is the 16X16 image.

The follow example shows how to display icons.

```
//----- PACKAGE -----
package biz.guidebee.example.drawing2d;

//----- IMPORTS -----
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import biz.guidebee.drawing.*;
import biz.guidebee.geometry.*;

//[----- MAIN CLASS -----]
//----- REVISIONS -----
// Date Name Tracking # Description
// -----
// 01SEP2007 James Shen Initial Creation
//-----
/**
 * This example show to to display icons.
 */
public class Icons extends MIDlet
{
 /**
 * The display object associated with the MIDlet.
 */
 private Display display;

 /**
 * The Canvas object.
 */
 private Canvas2D canvas;

 /**
 * The graphics object.
 */
 private Graphics2D graphics2D;

 Icon plane;
 Icon hand;

 public Icons() {

 display = Display.getDisplay(this);
 canvas = new Canvas2D(display);
 graphics2D = canvas.getGraphics();

 try{
 plane=new Icon(this.getClass().getResourceAsStream("/PLANE.ICO"));
 hand=new Icon(this.getClass().getResourceAsStream("/POINT04.ICO"));

 }catch (Exception e){
 System.out.println(e.getMessage());
 }

 }

 public void startApp() throws MIDletStateChangeException {
 display.setCurrent(canvas);
 //Clear the canvas with white color.
 }
}
```

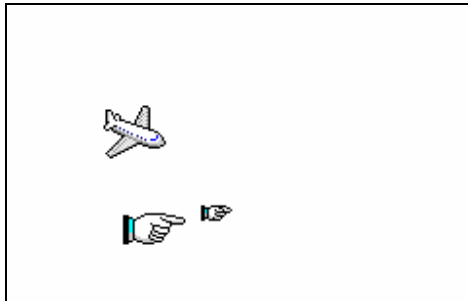
```

 graphics2D.clear(Color.WHITE);
 Image[] images=hand.toImages();
 graphics2D.drawIcon(plane,50,50);
 graphics2D.drawImage(images[0],60,100);
 graphics2D.drawImage(images[1],100,100);
 graphics2D.invalidate();
 }

 public void pauseApp() {
 }

 public void destroyApp(boolean unconditional)
 throws MIDletStateChangeException {
 canvas.flush();
 canvas = null;
 System.gc();
 }
}

```



### 5.3 Alpha blending of images

Alpha blending is supported by drawImage. The following example represents an image drawn on top of text. Alpha value specifies the transparency of the image. Click a Up or a Down to show more or less of the text through the image and make the image more or less transparent.

```

//----- PACKAGE -----
package biz.guidebee.example.drawing2d;

//----- IMPORTS -----
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

import biz.guidebee.drawing.*;
import biz.guidebee.geometry.*;

//[----- MAIN CLASS -----]
///
//----- REVISIONS -----
// Date Name Tracking # Description
// -----
// 01SEP2007 James Shen Initial Creation
///
/**
 * Click a Up or a Down to show more or less of the text through the image and
 * make the image more or less transparent.
 */
public class SeeThroughImage extends MIDlet implements CommandListener

```

```
{
 /**
 * The display object associated with the MIDlet.
 */
 private Display display;

 /**
 * The Canvas object.
 */
 private Canvas2D canvas;

 /**
 * The graphics object.
 */
 private Graphics2D graphics2D;

 private Image bi;

 int w, h;
 int offX,offY;

 int alpha=128;

 FontEx font=FontEx.getSystemFont();

 int fontSize=24;
 Pen pen=new Pen(Color.RED,2);
 char[] message="Java 2D is great!".toCharArray();
 int widthOfMessage=0;

 Command cmdAlphaUp=new Command("a Up",Command.SCREEN,1);
 Command cmdAlphaDown=new Command("a Down",Command.SCREEN,1);

 public void commandAction(Command c,
 Displayable d){
 if(c==cmdAlphaUp){
 alpha+=16;
 if(alpha>255) alpha=255;
 redraw();
 }else if(c==cmdAlphaDown){
 alpha-=16;
 if(alpha<0) alpha=0;
 redraw();
 }
 }

 public SeeThroughImage() {

 display = Display.getDisplay(this);
 canvas = new Canvas2D(display);
 graphics2D = canvas.getGraphics();

 try{
 bi=Image.createImage(this.getClass().
 getResourceAsStream("/duke_skateboard.jpg"));
 w = bi.getWidth();
 h = bi.getHeight();
 }catch (Exception e){
 System.out.println(e.getMessage());
 }
 }
}
```

```

 widthOfMessage=font.charsWidth(fontSize,message,0,message.length,
 FontEx.TEXT_DIR_LR);
 offX=(canvas.getWidth()-w)/2;
 offY=(canvas.getHeight()-h)/2;
 graphics2D.setDefaultPen(pen);
 canvas.addCommand(cmdAlphaUp);
 canvas.addCommand(cmdAlphaDown);
 canvas.setCommandListener(this);
 }

 void redraw(){
 graphics2D.clear(Color.WHITE);
 graphics2D.drawChars(font,fontSize,message,0,message.length,
 offX+(w-widthOfMessage)/2,offY+h/2);
 graphics2D.drawImage(bi,
 offX, offY,
 0xFFFF00FF,alpha);
 graphics2D.invalidate();
 }
 public void startApp() throws MIDletStateChangeException {
 display.setCurrent(canvas);
 //Clear the canvas with white color.

 redraw();
 }

 public void pauseApp() {
 }

 public void destroyApp(boolean unconditional)
 throws MIDletStateChangeException {
 canvas.flush();
 canvas = null;
 System.gc();
 }
}

```



## 5.4 Draw transparent images

When draw images on screen, a color value can be specified as the transparent color. Pixels with that color value will not be drawn when render the image on the canvas allow underneath picture be seen through.

The following example shows how to draw transparent images.

```
//----- PACKAGE -----
package biz.guidebee.example.drawing2d;

//----- IMPORTS -----
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

import biz.guidebee.drawing.*;
import biz.guidebee.geometry.*;

//[----- MAIN CLASS -----]
//----- REVISIONS -----
// Date Name Tracking # Description
// -----
// 01SEP2007 James Shen Initial Creation
//-----
/**
 * This example shows how to draw transparent images.
 */
public class TransparentImages extends MIDlet
{
 /**
 * The display object associated with the MIDlet.
 */
 private Display display;

 /**
 * The Canvas object.
 */
 private Canvas2D canvas;

 /**
 * The graphics object.
 */
 private Graphics2D graphics2D;

 Image backgroundImage;

 Image shape;

 Image guidebee;

 public TransparentImages() {

 display = Display.getDisplay(this);
 canvas = new Canvas2D(display);
 graphics2D = canvas.getGraphics();
 try{
 backgroundImage=Image.createImage(this.getClass().
 getResourceAsStream("/garden.png"));
 shape=Image.createImage(this.getClass().
```

```
 getResourceAsStream("/shp.png"));
 guidebee=Image.createImage(this.getClass().
 getResourceAsStream("/guidebee.png"));

 }catch (Exception e){
 System.out.println(e.getMessage());
 }
}

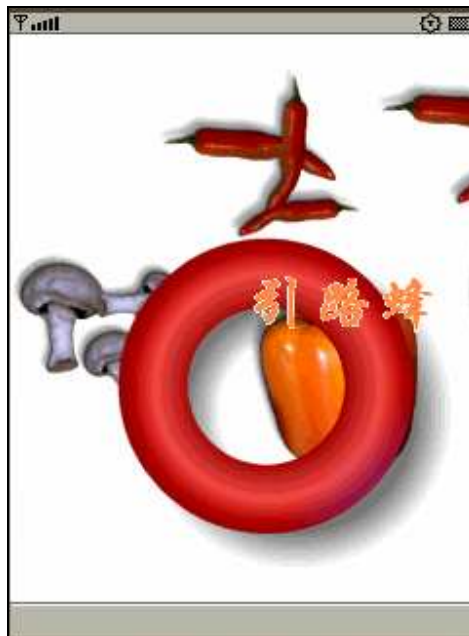
public void startApp() throws MIDletStateChangeException {
 display.setCurrent(canvas);
 //Clear the canvas with white color.
 graphics2D.clear(Color.WHITE);
 graphics2D.setBackgroundImage(backgroundImage);
 graphics2D.drawImage(shape,50,100);
 int []transpency=new int[1];
 guidebee.getRGB(transpency,0,1,0,0,1,1);

 graphics2D.drawImage(guidebee,120,120,transpency[0]);

 graphics2D.invalidate();
}

public void pauseApp() {
}

public void destroyApp(boolean unconditional)
throws MIDletStateChangeException {
 canvas.flush();
 canvas = null;
 System.gc();
}
}
```



## 5.5 Zoom in or Zoom out images

Class Graphics2D has one method zoomImage can be used to zoom in/zoom out a image.

The following example shows how to zoom in/zoom out images.

```
//----- PACKAGE -----
package biz.guidebee.example.drawing2d;

//----- IMPORTS -----
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

import biz.guidebee.drawing.*;
import biz.guidebee.geometry.*;

//[----- MAIN CLASS -----]
//----- REVISIONS -----
// Date Name Tracking # Description
// -----
// 01SEP2007 James Shen Initial Creation
//-----
/**
 * This example shows how to zoom in/zoom out images.
 */
public class ZoomImage extends MIDlet implements CommandListener
{
 /**
 * The display object associated with the MIDlet.
 */
 private Display display;

 /**
 * The Canvas object.
 */
 private Canvas2D canvas;

 /**
 * The graphics object.
 */
 private Graphics2D graphics2D;

 Image tomcat=null;
 Image tomcatNew=null;
 int w, h;
 int offX,offY;
 int zoomWidth=0;
 Command cmdZoomIn=new Command("Zoom In",Command.SCREEN,1);
 Command cmdZoomOut=new Command("Zoom Out",Command.SCREEN,1);

 public void commandAction(Command c,
 Displayable d){
 if(c==cmdZoomIn){
 zoomWidth+=10;
 if(zoomWidth>100) zoomWidth=100;
 tomcatNew=graphics2D.zoomImage(tomcat,w+zoomWidth,h+zoomWidth);
 redraw();
 }else if(c==cmdZoomOut){
 zoomWidth-=10;
 if(zoomWidth+w<10) zoomWidth=10-w;
 tomcatNew=graphics2D.zoomImage(tomcat,w+zoomWidth,h+zoomWidth);
 }
 }
}
```

```

 redraw();
 }
}
void redraw(){
 graphics2D.clear(Color.WHITE);
 int w1 = tomcatNew.getWidth();
 int h1 = tomcatNew.getHeight();
 offX=(canvas.getWidth()-w1)/2;
 offY=(canvas.getHeight()-h1)/2;
 graphics2D.drawImage(tomcatNew,offX,offY);
 graphics2D.invalidate();
}

public ZoomImage() {
 display = Display.getDisplay(this);
 canvas = new Canvas2D(display);
 graphics2D = canvas.getGraphics();
 try{
 tomcat=Image.createImage(this.getClass().
 getResourceAsStream("/tom.jpg"));
 tomcatNew=Image.createImage(this.getClass().
 getResourceAsStream("/tom.jpg"));
 w = tomcat.getWidth();
 h = tomcat.getHeight();
 }catch (Exception e){
 System.out.println(e.getMessage());
 }
 offX=(canvas.getWidth()-w)/2;
 offY=(canvas.getHeight()-h)/2;
 canvas.addCommand(cmdZoomIn);
 canvas.addCommand(cmdZoomOut);
 canvas.setCommandListener(this);
}

public void startApp() throws MIDletStateChangeException {
 display.setCurrent(canvas);
 redraw();
}

public void pauseApp() {
}

public void destroyApp(boolean unconditional)
throws MIDletStateChangeException {
 canvas.flush();
 canvas = null;
 System.gc();
}
}

```



## 6.0 Advanced Topics in Java2D

This section shows how to use Graphics2D to display graphics with fancy outline and fill styles, transform graphics when they are rendered, and how to create complex Shape objects by combining simple ones and how to detect when the user clicks on a displayed graphics primitive.

### 6.1 Transforming Shapes, Text, and Images

You can modify the transform attribute in the Graphics2D context to move, rotate, scale, and shear graphics primitives when they are rendered. The transform attribute is defined by an instance of the AffineTransform class. An affine transform is a transformation such as translate, rotate, scale, or shear in which parallel lines remain parallel even after being transformed.

The Graphics2D class provides several methods for changing the transform attribute. You can construct a new AffineTransform and change the Graphics2D transform attribute by calling transform.

AffineTransform defines the following factory methods to make it easier to construct new transforms:

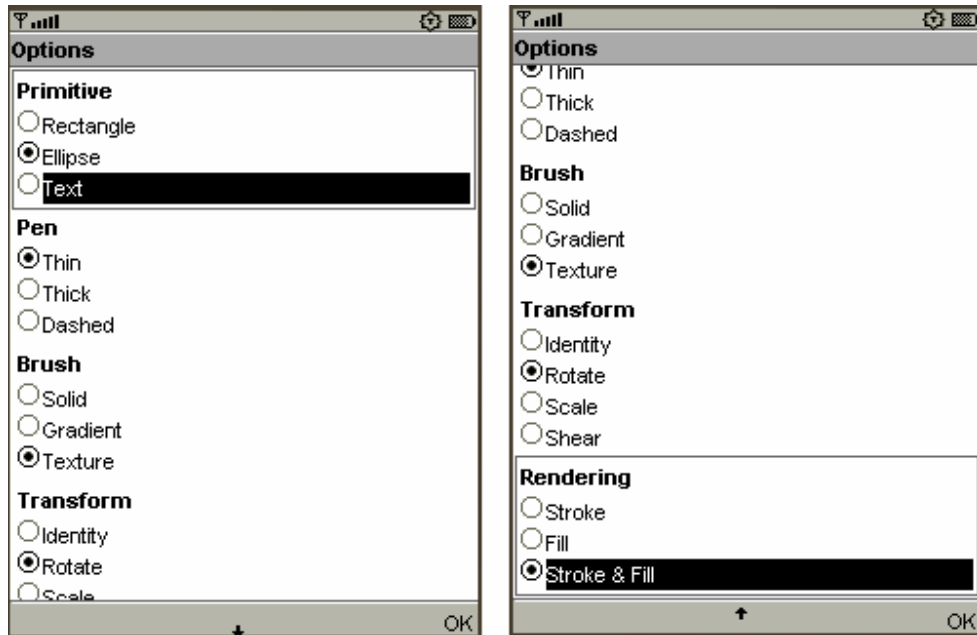
- getRotateInstance
- getScaleInstance
- getShearInstance
- getTranslateInstance

Alternatively you can use one of the Graphics2D transformation methods to modify the current transform. When you call one of these convenience methods, the resulting transform is concatenated with the current transform and is applied during rendering:

- rotate--to specify an angle of rotation in radians
- scale--to specify a scaling factor in the x and y directions
- shear--to specify a shearing factor in the x and y directions
- translate--to specify a translation offset in the x and y directions

You can also construct an AffineTransform object directly and concatenate it with the current transform by calling the transform method.

The following examples show to how to transform Shapes,Text , also can choose filling style and stroke style draw drawing Shapes or Text.



```

//----- PACKAGE -----
package biz.guidebee.example.drawing2d;

//----- IMPORTS -----
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

import biz.guidebee.drawing.*;
import biz.guidebee.geometry.*;

//[----- MAIN CLASS -----]
//----- REVISIONS -----
// Date Name Tracking # Description
// -----
// 01SEP2007 James Shen Initial Creation
//-----
/**
 * This example show transform of shapes ,text with different styles.
 */
public class Transform extends MIDlet implements CommandListener
{
 /**
 * The display object associated with the MIDlet.
 */
 public Display display;

 /**
 * The Canvas object.
 */
 public Canvas2D canvas;

 /**
 * The graphics object.
 */
 private Graphics2D graphics2D;

```

```
Command cmdOption=new Command("Option",Command.SCREEN,1);
Command cmdRedraw=new Command("Redraw",Command.SCREEN,1);

OptionForm optionForm=null;

AffineTransform at = new AffineTransform();
int w, h;
Shape shapes[] = new Shape[3];

boolean firstTime = true;
public static boolean no2D = false;
FontEx font=FontEx.getSystemFont();

public void commandAction(Command c,
 Displayable d){
 if(c==cmdOption){
 display.setCurrent(optionForm);
 }else if(c==cmdRedraw){
 redraw();
 }
}

public Transform() {

 display = Display.getDisplay(this);
 canvas = new Canvas2D(display);
 graphics2D = canvas.getGraphics();
 canvas.addCommand(cmdOption);
 canvas.addCommand(cmdRedraw);
 canvas.setCommandListener(this);
 graphics2D.setAffineTransform(new AffineTransform());
 if(optionForm==null){
 optionForm=new OptionForm(this,"Options");
 }

 shapes[0] = new Rectangle(0, 0, 100, 100);
 shapes[1] = new EllipseD(0.0, 0.0, 100.0, 100.0);
 shapes[2] = font.getGlyphOutline(96,"TEXT".toCharArray(),0,4,0,0,0);
}

public void setTrans(int transIndex) {
 // Sets the AffineTransform.
 switch (transIndex) {
 case 0 : at.setToIdentity();
 at.translate(w/2, h/2); break;
 case 1 : at.rotate(Math.toRadians(45)); break;
 case 2 : at.scale(0.5, 0.5); break;
 case 3 : at.shear(0.5, 0.0); break;
 }
}

public void redraw(){
 graphics2D.clear(Color.WHITE);

 if (!Transform.no2D) {

 w = canvas.getWidth();
 h = canvas.getHeight();

 setTrans(optionForm.transformIndex);
 }
}
```

```
graphics2D.setAffineTransform(at);

// Initialize the transform.
if (firstTime) {
 at.setToIdentity();
 at.translate(w/2, h/2);
 firstTime = false;
}

// Sets the Stroke.
Pen pen=null;

switch (optionForm.penIndex) {
case 0 :
 pen=new Pen(Color.BLACK,1);
 break;
case 1 :
 pen=new Pen(Color.BLACK,8);
 break;
case 2 :
 {
 int dash[] = {10,10};
 pen=new Pen(Color.BLACK,1,Pen.CAP_BUTT,Pen.JOIN_MITER,10,dash,0);
 }
 break;
}

// Sets the Paint.
Brush brush=null;

switch (optionForm.brushIndex) {
case 0 :
 brush=new SolidBrush(Color.BLUE);
 break;
case 1 :
 {
 int []fractions=new int[]{13,242};
 Color []colors=new Color[]{new Color(0xffff6600),
 new Color(0xffffffff66)};
 brush=new LinearGradientBrush(50,50,150,125,
 fractions,colors,Brush.REFLECT);
 }
 break;
case 2 :
 try{
 brush=new TextureBrush(Image.createImage(this.getClass().
 getResourceAsStream("/brick.png")));
 }
 catch(Exception e){}
 break;
}

// Sets the Shape.
Shape shape = shapes[optionForm.primitiveIndex];
Rectangle r = shape.getBounds();

AffineTransform toCenterAt = new AffineTransform();
toCenterAt.concatenate(at);
toCenterAt.translate(-(r.width/2), -(r.height/2));
```

```
 graphics2D.setAffineTransform(toCenterAt);

 // Sets the rendering method.
 switch (optionForm.renderingIndex) {
 case 0 :
 graphics2D.setDefaultPen(pen);
 graphics2D.draw(null,shape); break;
 case 1 :
 graphics2D.setDefaultBrush(brush);
 graphics2D.fill(null,shape); break;
 case 2 :
 graphics2D.setPenAndBrush(pen,brush);
 graphics2D.draw(null,shape); break;
 }
 graphics2D.invalidate();
 }

}

public void startApp() throws MIDletStateChangeException {
 display.setCurrent(canvas);
 //Clear the canvas with white color.
 redraw();
}

public void pauseApp() {
}

public void destroyApp(boolean unconditional)
throws MIDletStateChangeException {
 canvas.flush();
 canvas = null;
 System.gc();
}
}

//Option Form
/*
 * OptionForm.java
 */

package biz.guidebee.example.drawing2d;

import javax.microedition.lcdui.*;

public class OptionForm extends Form implements CommandListener{

 public static int PRIMITIVE_RECTANGLE=0;
 public static int PRIMITIVE_ELLIPSE=1;
 public static int PRIMITIVE_TEXT=2;

 public static int PEN_THIN=0;
 public static int PEN_THICK=1;
 public static int PEN_DASHED=2;

 public static int BRUSH_SOLID=0;
 public static int BRUSH_GRADIENT=1;
 public static int BRUSH_TEXTURE=2;

 public static int TRANSFORM_IDENTITY=0;
```

```
public static int TRANSFORM_ROTATE=1;
public static int TRANSFORM_SCALE=2;
public static int TRANSFORM_SHEAR=3;

public static int RENDERING_STROKE=0;
public static int RENDERING_FILL=1;
public static int RENDERING_STROKE_AND_FILL=2;

public int primitiveIndex;
public int penIndex;
public int brushIndex;
public int transformIndex;
public int renderingIndex;

ChoiceGroup cgPrimitive;
ChoiceGroup cgPen;
ChoiceGroup cgBrush;
ChoiceGroup cgTransform;
ChoiceGroup cgRendering;

Transform mainWindow=null;

Command cmdOK=new Command("OK",Command.OK,1);

public void commandAction(Command c,
 Displayable d){
 if(c==cmdOK){
 primitiveIndex=cgPrimitive.getSelectedIndex();
 penIndex=cgPen.getSelectedIndex();
 brushIndex=cgBrush.getSelectedIndex();
 transformIndex=cgTransform.getSelectedIndex();
 renderingIndex=cgRendering.getSelectedIndex();
 mainWindow.redraw();
 mainWindow.display.setCurrent(mainWindow.canvas);
 }
}

public OptionForm(Transform main,String title) {
 super(title);
 mainWindow=main;
 cgPrimitive =new ChoiceGroup("Primitive",ChoiceGroup.EXCLUSIVE);
 cgPrimitive.append("Rectangle",null);
 cgPrimitive.append("Ellipse",null);
 cgPrimitive.append("Text",null);
 this.append(cgPrimitive);

 cgPen =new ChoiceGroup("Pen",ChoiceGroup.EXCLUSIVE);
 cgPen.append("Thin",null);
 cgPen.append("Thick",null);
 cgPen.append("Dashed",null);
 this.append(cgPen);

 cgBrush =new ChoiceGroup("Brush",ChoiceGroup.EXCLUSIVE);
 cgBrush.append("Solid",null);
 cgBrush.append("Gradient",null);
 cgBrush.append("Texture",null);
 this.append(cgBrush);

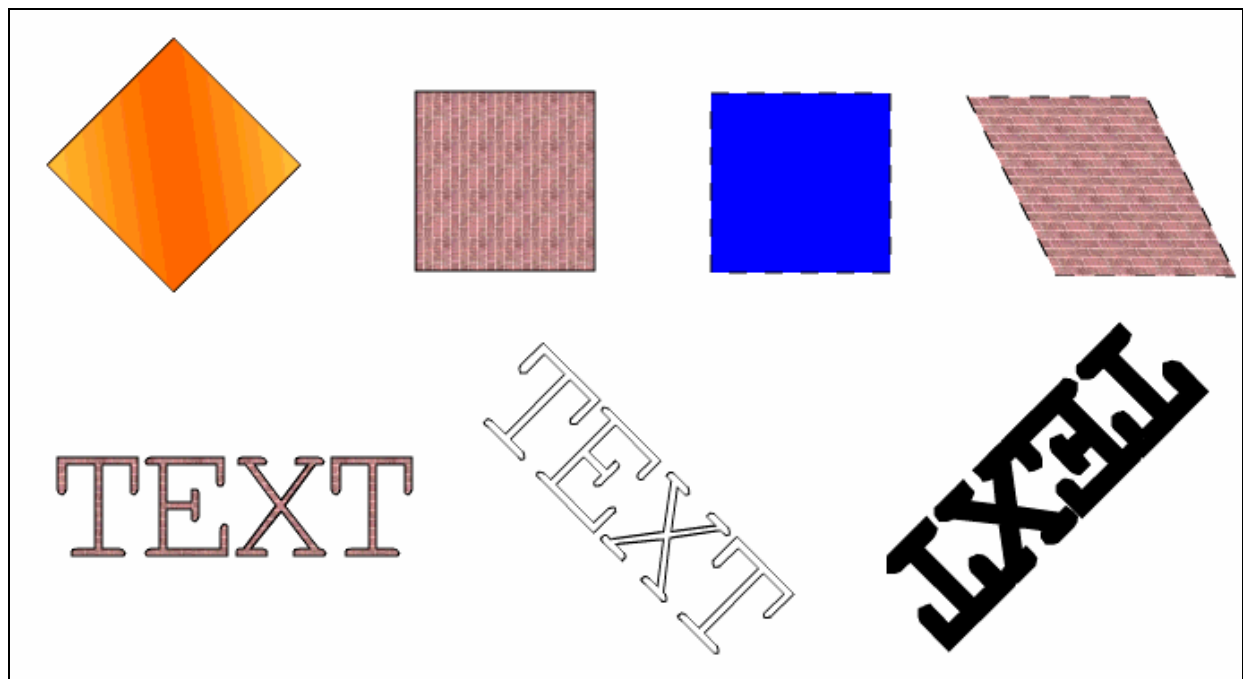
 cgTransform =new ChoiceGroup("Transform",ChoiceGroup.EXCLUSIVE);
 cgTransform.append("Identity",null);
 cgTransform.append("Rotate",null);
```

```
cgTransform.append("Scale",null);
cgTransform.append("Shear",null);
this.append(cgTransform);

cgRendering =new ChoiceGroup("Rendering",ChoiceGroup.EXCLUSIVE);
cgRendering.append("Stroke",null);
cgRendering.append("Fill",null);
cgRendering.append("Stroke & Fill",null);
this.append(cgRendering);

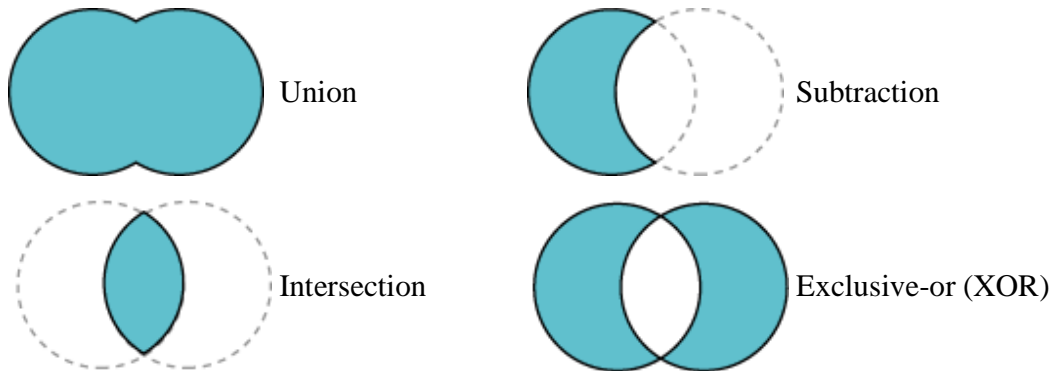
this.addCommand(cmdOK);
this.setCommandListener(this);

}
}
```



## 6.2 Constructing Complex Shapes from Geometry Primitives

Constructive area geometry (CAG) is the process of creating new geometric shapes by performing boolean operations on existing ones. In the Java 2D™ API the Area class implements the Shape interface and supports the following boolean operations.



In the following example Area objects construct a pear shape from several ellipses.

```
//----- PACKAGE -----
package biz.guidebee.example.drawing2d;

//----- IMPORTS -----
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

import biz.guidebee.drawing.*;
import biz.guidebee.geometry.*;

//[----- MAIN CLASS -----]
//----- REVISIONS -----
// Date Name Tracking # Description
// -----
// 01SEP2007 James Shen Initial Creation
//-----
/**
 * In this example Area objects construct a pear shape from several ellipses.
 */
public class Pear extends MIDlet
{
 /**
 * The display object associated with the MIDlet.
 */
 private Display display;

 /**
 * The Canvas object.
 */
 private Canvas2D canvas;

 /**
 * The graphics object.
 */
 private Graphics2D graphics2D;

 EllipseD circle, oval, leaf, stem;
 Area circ, ov, leaf1, leaf2, st1, st2;

 public void init() {
```

```
 circle = new EllipseD();
 oval = new EllipseD();
 leaf = new EllipseD();
 stem = new EllipseD();
 circ = new Area(circle);
 ov = new Area(oval);
 leaf1 = new Area(leaf);
 leaf2 = new Area(leaf);
 st1 = new Area(stem);
 st2 = new Area(stem);
 }

 public Pear() {

 display = Display.getDisplay(this);
 canvas = new Canvas2D(display);
 graphics2D = canvas.getGraphics();
 }

 public void startApp() throws MIDletStateChangeException {
 display.setCurrent(canvas);
 //Clear the canvas with white color.
 init();
 graphics2D.clear(Color.WHITE);

 int w = canvas.getWidth();
 int h = canvas.getHeight();
 double ew = w/2;
 double eh = h/2;
 SolidBrush brush=new SolidBrush(Color.GREEN);
 graphics2D.setDefaultBrush(brush);

 // Creates the first leaf by filling the
 //intersection of two Area objects created from an ellipse.
 leaf.setFrame(ew-16, eh-29, 15.0, 15.0);
 leaf1 = new Area(leaf);
 leaf.setFrame(ew-14, eh-47, 30.0, 30.0);
 leaf2 = new Area(leaf);
 leaf1.intersect(leaf2);
 graphics2D.fill(null,leaf1);

 // Creates the second leaf.
 leaf.setFrame(ew+1, eh-29, 15.0, 15.0);
 leaf1 = new Area(leaf);
 leaf2.intersect(leaf1);
 graphics2D.fill(null,leaf2);

 brush=new SolidBrush(Color.BLACK);
 graphics2D.setDefaultBrush(brush);

 // Creates the stem by filling the Area resulting
 //from the subtraction of two Area objects created from an ellipse.
 stem.setFrame(ew, eh-42, 40.0, 40.0);
 st1 = new Area(stem);
 stem.setFrame(ew+3, eh-47, 50.0, 50.0);
 st2 = new Area(stem);
 st1.subtract(st2);
 graphics2D.fill(null,st1);

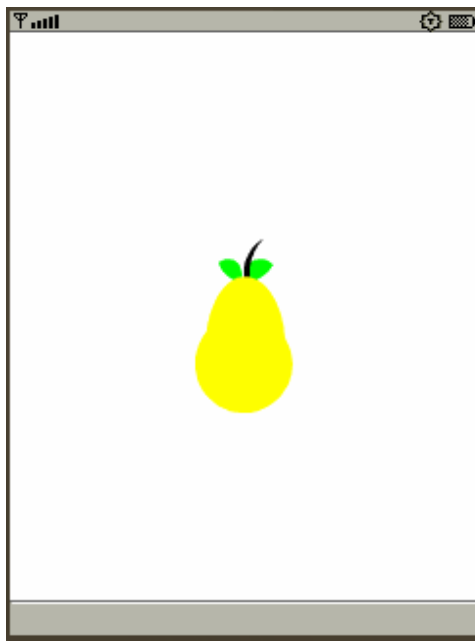
 brush=new SolidBrush(Color.YELLOW);
 graphics2D.setDefaultBrush(brush);
 }
}
```

```
// Creates the pear itself by filling the Area resulting
//from the union of two Area objects created by two different ellipses.
circle.setFrame(ew-25, eh, 50.0, 50.0);
oval.setFrame(ew-19, eh-20, 40.0, 70.0);
circ = new Area(circle);
ov = new Area(oval);
circ.add(ov);
graphics2D.fill(null,circ);

graphics2D.invalidate();
}

public void pauseApp() {
}

public void destroyApp(boolean unconditional)
throws MIDletStateException {
 canvas.flush();
 canvas = null;
 System.gc();
}
}
```



### 6.3 Hit Path Test

Graphics2D also provides one method `hitPath` to check whether a given point intersects with a path.

```
public final boolean hitPath(int x,
 int y,
 Shape path,
 AffineTransform matrix)
```

Checks whether or not the specified point (x, y) intersects (hits) the given path.

The following example shows how to use the hitPath method, this Midlet needs device which supports touch screen, it allows user to move the path around.

```
//----- PACKAGE -----
package biz.guidebee.example.tiny2d;

//----- IMPORTS -----
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

import biz.guidebee.drawing.*;
import biz.guidebee.geometry.*;

//[----- MAIN CLASS -----]
//----- REVISIONS -----
// Date Name Tracking # Description
// -----
// 01SEP2007 James Shen Initial Creation
//-----
/**
 * This example displays how to implement hit test.
 */
public class HitPath extends MIDlet implements Runnable {
 /**
 * The display object associated with the MIDlet.
 */
 private Display display;

 /**
 * The Canvas object.
 */
 private HitPathCanvas canvas;

 /**
 * The graphics object.
 */
 private Graphics2D graphics2D;

 /* The oval outline */
 GeneralPath path;
 /* The matrix */
 AffineTransform matrix = new AffineTransform();
 /** Colors */
 RadialGradientBrush brush;

 /* The thread. */
 private Thread thread;
 boolean drawn;

 FontEx font=FontEx.getSystemFont();

 Pen pen;

 Pen pen1;

 //static char[] pathdata =
 // "M 60 20 Q -40 70 60 120 Q 160 70 60 20 z".toCharArray();
 static char[]pathdata="M 186 90L 178 95L 184 127L 165 132L 147 103L 110 103L 122
177Q 116 180 102 179Q 94 178 90 168Q 86 155 66 103L 40 103Q 12 103 0 90Q 12 76 40
76L 66 76L 90 12Q 94 2 102 1Q 116 0 122 3L 110 77L 147 77L 165 47L 184 53L 178 85L
```

```
186 90L 186 90Z".toCharArray();

static char[]hitYes="Yes".toCharArray();
static char[]HitNo="No".toCharArray();
int counter=1;
public HitPath() {
 display = Display.getDisplay(this);
 path=Graphics2D.toPath(new String(pathdata));
 canvas = new HitPathCanvas(display,path,matrix);
 graphics2D = canvas.getGraphics();
 int []fractions=new int[]{13,128,255};
 Color []colors=new Color[]{new Color(0xffff6600),new Color(0xfffff66),
 new Color(0xffff6600)};
 brush=new RadialGradientBrush(90,100,50,fractions,colors);
 pen=new Pen(brush,8);

}

public void start() {
 if (thread == null) {
 thread = new Thread(this);
 thread.start();
 }
}

public void stop() {
 thread = null;
}

public void run() {
 Thread me = Thread.currentThread();
 while (thread == me) {
 synchronized(canvas.lock) {
 graphics2D.clear(Color.BLACK);
 graphics2D.setAffineTransform(matrix);
 graphics2D.draw(pen,path);

 }

 graphics2D.invalidate();

 try {
 thread.sleep(25);
 } catch (Exception e) { break; }
 }
 thread = null;
}

public void startApp() throws MIDletStateChangeException {
 display.setCurrent(canvas);
 if (thread == null) {
 thread = new Thread(this);
 thread.start();
 }
}

public void pauseApp() {
}

public void destroyApp(boolean unconditional)
```

```

 throws MIDletStateChangeException {
 thread = null;
 canvas = null;
 System.gc();
 }
}

```

HitPathCanvas code.

```

//----- PACKAGE -----
package biz.guidebee.example.tiny2d;

//----- IMPORTS -----
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

import biz.guidebee.drawing.*;
import biz.guidebee.geometry.*;

public class HitPathCanvas extends Canvas2D{
 int pressedX;
 int pressedY;
 int draggedX;
 int draggedY;
 public Object lock = new Object();

 /** The hit flag */
 public boolean hit;
 /* The oval outline */
 GeneralPath path;
 /* The matrix */
 AffineTransform matrix;

 public HitPathCanvas(Display display, GeneralPath path, AffineTransform matrix)
 {
 super(display);
 this.path = path;
 this.matrix = matrix;
 hit=false;
 }

 /**
 * Invoked when the pointer has been pressed in the canvas.
 */
 protected void pointerPressed(int x, int y)
 {
 synchronized(lock)
 {
 // System.out.println("pointerPressed " + " " + x + " " + y);
 pressedX = x;
 pressedY = y;
 draggedX = pressedX;
 draggedY = pressedY;
 hit = graphics2D.hitPath(pressedX, pressedY, path, null);

 System.out.println("hit " + hit);
 }
 }
}
/**

```

```
 * Invoked when the pointer has been released in the canvas.
 */
 protected void pointerReleased(int x, int y)
 {
 synchronized(lock)
 {
 // System.out.println("pointerReleased " + " " + x + " " + y);
 //textMessage="Released ";

 if(hit)
 {
 matrix.translate(x-pressedX ,y-pressedY);
 }
 }
 }
}

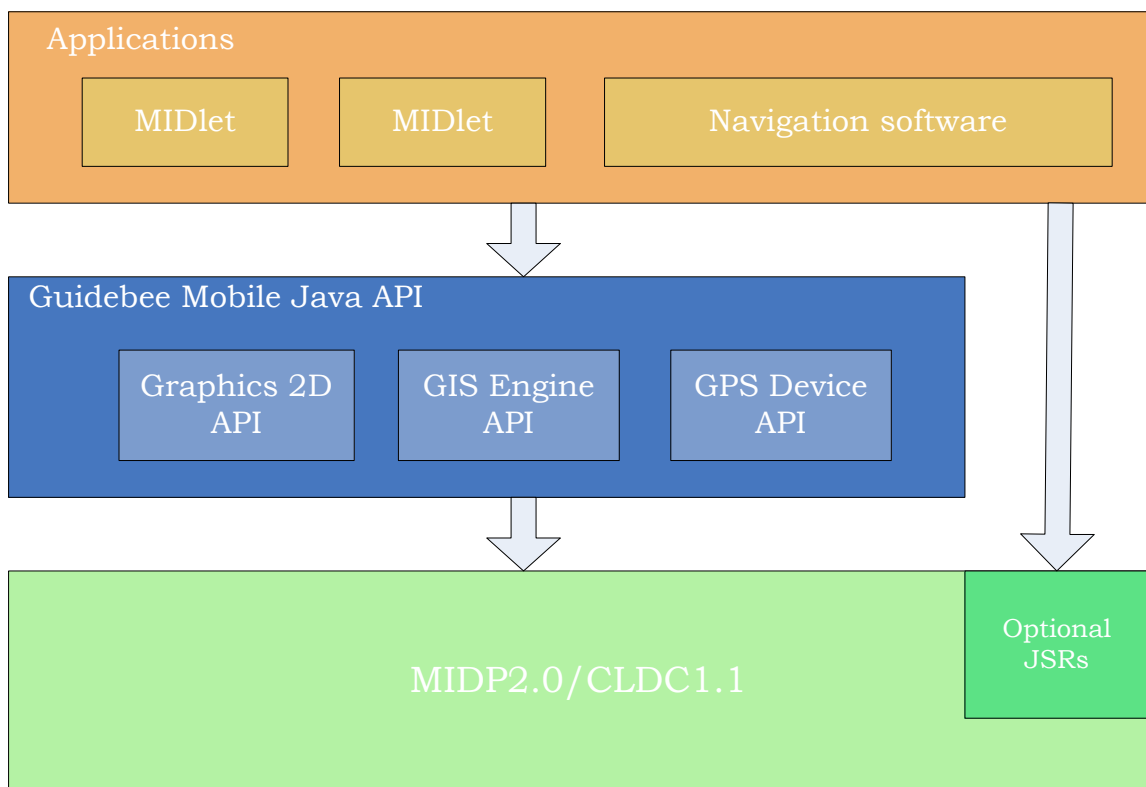
/**
 * Invoked when the pointer is pressed within the canvas and then
 * dragged.
 */
protected void pointerDragged(int x, int y)
{
 synchronized(lock)
 {
 draggedX = x;
 draggedY = y;
 }
}
}
```



## 7.0 About Guidebee Graphics 2D API

Guidebee Graphics 2D APIs was originally designed as part of navigation software, which developed based on the following three core APIs.

- Graphics 2D API  
APIs support 2-dimensional graphics drawing. Shape, Texts, Images.
- GIS Engine API  
Provides Geographics Information Services either online or offline. Supports Map-Matching, Spatial query, Routing, Navigation etc.
- GPS Device API  
Communicate with different type of GPS devices to get location information.



But the Graphics 2D API has no dependency on other two APIs; it can be used standalone with other MIDlets beside navigation software. In order to let the Graphics 2D API work properly it requires CLDC1.1 which has float support. That's because the Graphics 2D API includes float operation to achieve more precise calculation results.

You can obtain the Graphics 2D API with two different licenses: trial and commercial license. For more information please visit [www.guidebee.biz](http://www.guidebee.biz)

## 8.0 Licence

"This product includes TinyLine software licensed from Andrew Girow 2002- 2007 (<http://www.tinyline.com/>)."

Currently part of biz.guidebee.drawing package was developed on top of TinyLine library and has obtained the commercial licence of Tinyline 2D library. For more information about tinyline licence please refer to [http://tinyline.com/store/TINYLINE\\_LICENSE.TXT](http://tinyline.com/store/TINYLINE_LICENSE.TXT)

Note: In future release of Guidebee Graphics2D API, TinyLine may be removed without pre-notice. The APIs of the Graphics 2D library keeps same interface, developers don't need to bother to change your code to use the new library.

To use Guidebee Graphics2D library, you don't necessary obtain any licence from Tinyline, but you need get a licence, either trial or commercial licence from James Shen (Guidebee Biz.).

Licence from Guidebee Biz. Includes two parts:

- The licence key

Gives the product name "Graphcis2D" and licence keys, six long integers. To correctly use this library, you need to call LicenceManager to add correctly licence to the LicenceManager.

- And the licence file.

The file name is guidebee.lic, which need to put in the root directory of the resource with the MIDlet jar file.

If develops with Wireless tool kit, the licence file needs to put in the res directory. If develops with Netbean, the licence file needs to put in the src directory.

```
package biz.guidebee.example;

import javax.microedition.midlet.*;
import biz.guidebee.licence.*;

public class LicenceExample extends MIDlet{
 /**
 * The display object associated with the MIDlet.
 */
 private Display display;

 /**
 * The Canvas object.
 */
 private Canvas2D canvas;

 /**
 * The graphics object.
 */
 private Graphics2D graphics2D;
```

```

public LicenceExample () {
 try{
 LicenceManager licenceManager=LicenceManager.getInstance(this);
 long keys[]= {-0x77f0a31e232a2957L, 0x142abbb790d9022cL,
 -0x4161df11675bd95cL,-0x53e3e06a081e6286L,
 -0x4078bd366ba7c517L, 0x6782da059093574bL};
 licenceManager.addLicence("Graphics2D",keys);
 display = Display.getDisplay(this);
 canvas = new Canvas2D(display);
 graphics2D = canvas.getGraphics();
 }catch(InvalidLicenceException e){
 System.out.println(e.getMessage());
 }
}
}

```

LicenceManager locates in package biz.guidebee.licence which is not part of Graphics 2D API.

It has two static methods.

```

public static LicenceManager getInstance(MIDlet midlet)
public void addLicence(String appName,long[]keys)
 throws InvalidLicenceException

```

getInstance returns an instance of LicenceManager ,it takes the MIDlet instance which uses the library as the input parameter.

addLicence add licence to the LicenceManager, it takes the appName ,in this case "Graphics2D" and the licence key.

If the licence key is invalid ,missing or the guidebee.lic is invalid,missing or placed in wrong place, InvalidLicenceException will be thrown, or you forget to add the code to add the licence to the LicenceManager, the following code wont work.

```

canvas = new Canvas2D(display);

```

**will return NULL!**

In this case, you cannot use most of the core API of Graphics2D library.

## 9.0 References

1. The Java Tutorial Trail: 2D Graphics  
<http://java.sun.com/docs/books/tutorial/2d/TOC.html>
2. Java Standard Version 6.0 Java doc <http://java.sun.com>
3. MSDN Library for Visual Studio .Net 2003
4. TinyLine 2D Programming Guide <http://www.tinyline.com>