Common Text Layout Operations

This chapter describes some common text layout operations and shows how they can be performed using Core Text. The following operations with code listings are included in this chapter:

- Laying Out a Paragraph
- Simple Text Label
- Columnar Layout
- Manual Line Breaking
- Applying a Paragraph Style
- Displaying Text in a Nonrectangular Region

Laying Out a Paragraph

One of the most common operations in typesetting is laying out a multiline paragraph within an arbitrarily sized rectangular area. Core Text makes this operation easy, requiring only a few lines of Core Text–specific code. To lay out the paragraph, you need a graphics context to draw into, a rectangular path to provide the area where the text is laid out, and an attributed string. Most of the code in this example is required to create and initialize the context, path, and string. After that is done, Core Text requires only three lines of code to do the layout.

The code in Listing 2-1 shows how a paragraph is laid out. This code could reside in the `drawRect:` method of a `UIView` subclass (an `NSView` subclass in OS X).

Listing 2-1 Typesetting a simple paragraph

```
// Initialize a graphics context in iOS.
CGContextRef context = UIGraphicsGetCurrentContext();


// Flip the context coordinates, in iOS only.
CGContextTranslateCTM(context, 0, self.bounds.size.height);
CGContextScaleCTM(context, 1.0, -1.0);


// Initializing a graphic context in OS X is different:
// CGContextRef context =
//     (CGContextRef)[[NSGraphicsContext currentContext]
graphicsPort];


// Set the text matrix.
```

```
CGContextSetTextMatrix(context, CGAffineTransformIdentity);


// Create a path which bounds the area where you will be drawing text.
// The path need not be rectangular.
CGMutablePathRef path = CGPathCreateMutable();


// In this simple example, initialize a rectangular path.
CGRect bounds = CGRectMake(10.0, 10.0, 200.0, 200.0);
CGPathAddRect(path, NULL, bounds );


// Initialize a string.
CFStringRef textString = CFSTR("Hello, World! I know nothing in the
world that has as much power as a word. Sometimes I write one, and I
look at it, until it begins to shine.");


// Create a mutable attributed string with a max length of 0.
// The max length is a hint as to how much internal storage to
reserve.
// 0 means no hint.
CFMutableAttributedStringRef attrString =
        CFAttributedStringCreateMutable(kCFAllocatorDefault, 0);


// Copy the textString into the newly created attrString
CFAttributedStringReplaceString (attrString, CFRangeMake(0, 0),
        textString);


// Create a color that will be added as an attribute to the
attrString.
CGColorSpaceRef rgbColorSpace = CGColorSpaceCreateDeviceRGB();
CGFloat components[] = { 1.0, 0.0, 0.0, 0.8 };
CGColorRef red = CGColorCreate(rgbColorSpace, components);
CGColorSpaceRelease(rgbColorSpace);


// Set the color of the first 12 chars to red.
CFAttributedStringSetAttribute(attrString, CFRangeMake(0, 12),
        kCTForegroundColorAttributeName, red);


// Create the framesetter with the attributed string.
CTFramesetterRef framesetter =
        CTFramesetterCreateWithAttributedString(attrString);
CFRelease(attrString);
```

```
// Create a frame.
CTFrameRef frame = CTFramesetterCreateFrame(framesetter,
          CFRangeMake(0, 0), path, NULL);

// Draw the specified frame in the given context.
CTFrameDraw(frame, context);

// Release the objects we used.
CFRelease(frame);
CFRelease(path);
CFRelease(framesetter);
```

Simple Text Label

Another common typesetting operation is drawing a single line of text to use as a label for a user-interface element. In Core Text this requires only two lines of code: one to create the line object with a CFAttributedString and another to draw the line into a graphic context.

Listing 2-2 shows how this would be done in the `drawRect:` method of a `UIView` or `NSView` subclass. The listing omits initialization of the plain text string, font, and graphics context, operations shown in other listings in this document. It shows how to create an attributes dictionary and use it to create the attributed string. (Font creation is shown in Creating Font Descriptors and Creating a Font from a Font Descriptor.)

Listing 2-2 Typesetting a simple text label

```
CFStringRef string; CTFontRef font; CGContextRef context;
// Initialize the string, font, and context

CFStringRef keys[] = { kCTFontAttributeName };
CFTypeRef values[] = { font };

CFDictionaryRef attributes =
    CFDictionaryCreate(kCFAllocatorDefault, (const void**)&keys,
        (const void**)&values, sizeof(keys) / sizeof(keys[0]),
        &kCFTypeDictionaryKeyCallBacks,
        &kCFTypeDictionaryValueCallBacks);

CFAttributedStringRef attrString =
    CFAttributedStringCreate(kCFAllocatorDefault, string, attributes);
```

```
CFRelease(string);
CFRelease(attributes);

CTLineRef line = CTLineCreateWithAttributedString(attrString);

// Set text position and draw the line into the graphics context
CGContextSetTextPosition(context, 10.0, 10.0);
CTLineDraw(line, context);
CFRelease(line);
```

Columnar Layout

Laying out text in multiple columns is another common typesetting operation. Strictly speaking, Core Text itself only lays out one column at a time and does not calculate the column sizes or locations. You do those operations before calling Core Text to lay out the text within the path area you've calculated. In this sample, Core Text, in addition to laying out the text in each column, also provides the subrange within the text string for each column.

The `createColumnsWithColumnCount:` method in Listing 2-3 accepts as a parameter the number of columns to be drawn and returns an array of paths, one path for each column.

Listing 2-4 includes an implementation of the `drawRect:` method, which calls the local `createColumnsWithColumnCount` method, defined first in this listing. This code resides in a `UIView` subclass (an `NSView` subclass in OS X). The subclass includes an `attributedString` property, which is not shown here but whose accessor is called in this listing to return the attributed string to be laid out.

Listing 2-3 Dividing a view into columns

```
- (CFArrayRef)createColumnsWithColumnCount:(int)columnCount
{
    int column;

    CGRect* columnRects = (CGRect*)calloc(columnCount,
sizeof(*columnRects));
    // Set the first column to cover the entire view.
    columnRects[0] = self.bounds;

    // Divide the columns equally across the frame's width.
    CGFloat columnWidth = CGRectGetWidth(self.bounds) / columnCount;
    for (column = 0; column < columnCount - 1; column++) {
        CGRectDivide(columnRects[column], &columnRects[column],
```

```
                    &columnRects[column + 1], columnWidth,
CGRectMinXEdge);
    }

   // Inset all columns by a few pixels of margin.
    for (column = 0; column < columnCount; column++) {
        columnRects[column] = CGRectInset(columnRects[column], 8.0,
15.0);
    }

    // Create an array of layout paths, one for each column.
    CFMutableArrayRef array =
                    CFArrayCreateMutable(kCFAllocatorDefault,
                                         columnCount,
&kCFTypeArrayCallBacks);

    for (column = 0; column < columnCount; column++) {
        CGMutablePathRef path = CGPathCreateMutable();
        CGPathAddRect(path, NULL, columnRects[column]);
        CFArrayInsertValueAtIndex(array, column, path);
        CFRelease(path);
    }
    free(columnRects);
    return array;
}
```

Listing 2-4 Performing columnar text layout

```
// Override drawRect: to draw the attributed string into columns.
// (In OS X, the drawRect: method of NSView takes an NSRect parameter,
//  but that parameter is not used in this listing.)
- (void)drawRect:(CGRect)rect
{
    // Initialize a graphics context in iOS.
    CGContextRef context = UIGraphicsGetCurrentContext();

    // Flip the context coordinates in iOS only.
    CGContextTranslateCTM(context, 0, self.bounds.size.height);
    CGContextScaleCTM(context, 1.0, -1.0);

    // Initializing a graphic context in OS X is different:
    // CGContextRef context =
```

```
    //     (CGContextRef)[[NSGraphicsContext currentContext]
graphicsPort];


    // Set the text matrix.
    CGContextSetTextMatrix(context, CGAffineTransformIdentity);


    // Create the framesetter with the attributed string.
    CTFramesetterRef framesetter =
CTFramesetterCreateWithAttributedString(

(CFAttributedStringRef)self.attributedString);


    // Call createColumnsWithColumnCount function to create an array
of
    // three paths (columns).
    CFArrayRef columnPaths = [self createColumnsWithColumnCount:3];


    CFIndex pathCount = CFArrayGetCount(columnPaths);
    CFIndex startIndex = 0;
    int column;


    // Create a frame for each column (path).
    for (column = 0; column < pathCount; column++) {
        // Get the path for this column.
        CGPathRef path =
(CGPathRef)CFArrayGetValueAtIndex(columnPaths, column);


        // Create a frame for this column and draw it.
        CTFrameRef frame = CTFramesetterCreateFrame(
                                framesetter, CFRangeMake(startIndex, 0),
path, NULL);
        CTFrameDraw(frame, context);


        // Start the next frame at the first character not visible in
this frame.
        CFRange frameRange = CTFrameGetVisibleStringRange(frame);
        startIndex += frameRange.length;
        CFRelease(frame);


    }
    CFRelease(columnPaths);
    CFRelease(framesetter);
```

```
}
```

Manual Line Breaking

In Core Text, you usually don't need to do manual line breaking unless you have a special hyphenation process or a similar requirement. A framesetter performs line breaking automatically. Alternatively, Core Text enables you to specify exactly where you want each line of text to break. Listing 2-5 shows how to create a typesetter, an object used by the framesetter, and use the typesetter directly to find appropriate line breaks and create a typeset line manually. This sample also shows how to center a line before drawing.

This code could be in the `drawRect:` method of a `UIView` subclass (an `NSView` subclass in OS X). The listing does not show initialization of the variables used in the code.

Listing 2-5 Performing manual line breaking

```
double width; CGContextRef context; CGPoint textPosition;
CFAttributedStringRef attrString;

// Initialize those variables.


// Create a typesetter using the attributed string.

CTTypesetterRef typesetter =
CTTypesetterCreateWithAttributedString(attrString);


// Find a break for line from the beginning of the string to the given
width.

CFIndex start = 0;

CFIndex count = CTTypesetterSuggestLineBreak(typesetter, start,
width);


// Use the returned character count (to the break) to create the line.

CTLineRef line = CTTypesetterCreateLine(typesetter, CFRangeMake(start,
count));


// Get the offset needed to center the line.

float flush = 0.5; // centered

double penOffset = CTLineGetPenOffsetForFlush(line, flush, width);


// Move the given text drawing position by the calculated offset and
draw the line.

CGContextSetTextPosition(context, textPosition.x + penOffset,
textPosition.y);

CTLineDraw(line, context);
```

```
// Move the index beyond the line break.
start += count;
```

Applying a Paragraph Style

Listing 2-6 implements a function that applies a paragraph style to an attributed string. The function accepts as parameters the font name, point size, and a line spacing which increases or decreases the amount of space between lines of text. This function is called by the code in Listing 2-7, which creates a plain text string, uses the `applyParaStyle` function to make an attributed string with the given paragraph attributes, then creates a framesetter and frame, and draws the frame.

Listing 2-6 Applying a paragraph style

```
NSAttributedString* applyParaStyle(
                CFStringRef fontName , CGFloat pointSize,
                NSString *plainText, CGFloat lineSpaceInc){

    // Create the font so we can determine its height.
    CTFontRef font = CTFontCreateWithName(fontName, pointSize, NULL);

    // Set the lineSpacing.
    CGFloat lineSpacing = (CTFontGetLeading(font) + lineSpaceInc) * 2;

    // Create the paragraph style settings.
    CTParagraphStyleSetting setting;

    setting.spec = kCTParagraphStyleSpecifierLineSpacing;
    setting.valueSize = sizeof(CGFloat);
    setting.value = &lineSpacing;

    CTParagraphStyleRef paragraphStyle =
CTParagraphStyleCreate(&setting, 1);

    // Add the paragraph style to the dictionary.
    NSDictionary *attributes = [NSDictionary
dictionaryWithObjectsAndKeys:
                                (__bridge id)font,
(id)kCTFontNameAttribute,
                                (__bridge id)paragraphStyle,
                                (id)kCTParagraphStyleAttributeName,
```

```
nil];
    CFRelease(font);
    CFRelease(paragraphStyle);

    // Apply the paragraph style to the string to created the
attributed string.
    NSAttributedString* attrString = [[NSAttributedString alloc]
                                initWithString:(NSString*)plainText
                                attributes:attributes];

    return attrString;
}
```

In Listing 2-7, the styled string is used to create a framesetter. The code uses the framesetter to create a frame and draws the frame.

Listing 2-7 Drawing the styled paragraph

```
- (void)drawRect:(CGRect)rect {
    // Initialize a graphics context in iOS.
    CGContextRef context = UIGraphicsGetCurrentContext();

    // Flip the context coordinates in iOS only.
    CGContextTranslateCTM(context, 0, self.bounds.size.height);
    CGContextScaleCTM(context, 1.0, -1.0);

    // Set the text matrix.
    CGContextSetTextMatrix(context, CGAffineTransformIdentity);

    CFStringRef fontName = CFSTR("Didot Italic");
    CGFloat pointSize = 24.0;

    CFStringRef string = CFSTR("Hello, World! I know nothing in the world
that has
                                    as much power as a word. Sometimes I
write one,
                                    and I look at it, until it begins to
shine.");

    // Apply the paragraph style.
    NSAttributedString* attrString = applyParaStyle(fontName, pointSize,
string, 50.0);

    // Put the attributed string with applied paragraph style into a
```

```
framesetter.
    CTFramesetterRef framesetter =

CTFramesetterCreateWithAttributedString((CFAttributedStringRef)attrString);

    // Create a path to fill the View.
    CGPathRef path = CGPathCreateWithRect(rect, NULL);

    // Create a frame in which to draw.
    CTFrameRef frame = CTFramesetterCreateFrame(
                                    framesetter, CFRangeMake(0, 0), path,
NULL);

    // Draw the frame.
    CTFrameDraw(frame, context);
    CFRelease(frame);
    CGPathRelease(path);
    CFRelease(framesetter);
}
```

In OS X, the `NSView drawRect:` method receives an `NSRect` parameter, but the `CGPathCreateWithRect` function requires a `CGRect` parameter. So, you must convert the `NSRect` object to a `CGRect` object with the following function call:

```
CGRect myRect = NSRectToCGRect([self bounds]);
```

In addition, in OS X, you get the graphics context differently and you do not flip its coordinates, as shown in comments in Listing 2-7.

Displaying Text in a Nonrectangular Region

The hard part of displaying text in a nonrectangular region is to describe the non-rectangular path. The `AddSquashedDonutPath` function in Listing 2-8 returns a donut-shaped path. Once you have the path, simply call the usual Core Text functions to apply attributes and draw.

Listing 2-8 Displaying text in a nonrectangular path

```
// Create a path in the shape of a donut.
static void AddSquashedDonutPath(CGMutablePathRef path,
              const CGAffineTransform *m, CGRect rect)
{
    CGFloat width = CGRectGetWidth(rect);
```

```
    CGFloat height = CGRectGetHeight(rect);


    CGFloat radiusH = width / 3.0;
    CGFloat radiusV = height / 3.0;


    CGPathMoveToPoint( path, m, rect.origin.x, rect.origin.y + height
- radiusV);
    CGPathAddQuadCurveToPoint( path, m, rect.origin.x, rect.origin.y +
height,
                               rect.origin.x + radiusH, rect.origin.y
+ height);
    CGPathAddLineToPoint( path, m, rect.origin.x + width - radiusH,
                               rect.origin.y + height);
    CGPathAddQuadCurveToPoint( path, m, rect.origin.x + width,
                               rect.origin.y + height,
                               rect.origin.x + width,
                               rect.origin.y + height - radiusV);
    CGPathAddLineToPoint( path, m, rect.origin.x + width,
                               rect.origin.y + radiusV);
    CGPathAddQuadCurveToPoint( path, m, rect.origin.x + width,
rect.origin.y,
                               rect.origin.x + width - radiusH,
rect.origin.y);
    CGPathAddLineToPoint( path, m, rect.origin.x + radiusH,
rect.origin.y);
    CGPathAddQuadCurveToPoint( path, m, rect.origin.x, rect.origin.y,
                               rect.origin.x, rect.origin.y +
radiusV);
    CGPathCloseSubpath( path);


    CGPathAddEllipseInRect( path, m,
                            CGRectMake( rect.origin.x + width / 2.0 -
width / 5.0,
                            rect.origin.y + height / 2.0 - height /
5.0,
                            width / 5.0 * 2.0, height / 5.0 * 2.0));
}


// Generate the path outside of the drawRect call so the path is
calculated only once.
- (NSArray *)paths
{
    CGMutablePathRef path = CGPathCreateMutable();
    CGRect bounds = self.bounds;
```

```
    bounds = CGRectInset(bounds, 10.0, 10.0);
    AddSquashedDonutPath(path, NULL, bounds);

    NSMutableArray *result =
              [NSMutableArray
arrayWithObject:CFBridgingRelease(path)];
    return result;
}

- (void)drawRect:(CGRect)rect
{
    [super drawRect:rect];

    // Initialize a graphics context in iOS.
    CGContextRef context = UIGraphicsGetCurrentContext();

    // Flip the context coordinates in iOS only.
    CGContextTranslateCTM(context, 0, self.bounds.size.height);
    CGContextScaleCTM(context, 1.0, -1.0);

    // Set the text matrix.
    CGContextSetTextMatrix(context, CGAffineTransformIdentity);

    // Initialize an attributed string.
    CFStringRef textString = CFSTR("Hello, World! I know nothing in
the world that
    has as much power as a word. Sometimes I write one, and I look at
it,
    until it begins to shine.");

    // Create a mutable attributed string.
     CFMutableAttributedStringRef attrString =
                CFAttributedStringCreateMutable(kCFAllocatorDefault,
0);

    // Copy the textString into the newly created attrString.
    CFAttributedStringReplaceString (attrString, CFRangeMake(0, 0),
textString);

    // Create a color that will be added as an attribute to the
attrString.
    CGColorSpaceRef rgbColorSpace = CGColorSpaceCreateDeviceRGB();
    CGFloat components[] = { 1.0, 0.0, 0.0, 0.8 };
```

```
    CGColorRef red = CGColorCreate(rgbColorSpace, components);
    CGColorSpaceRelease(rgbColorSpace);

    // Set the color of the first 13 chars to red.
    CFAttributedStringSetAttribute(attrString, CFRangeMake(0, 13),
                                   kCTForegroundColorAttributeName,
red);

    // Create the framesetter with the attributed string.
    CTFramesetterRef framesetter =
CTFramesetterCreateWithAttributedString(attrString);

    // Create the array of paths in which to draw the text.
    NSArray *paths = [self paths];

    CFIndex startIndex = 0;

    // In OS X, use NSColor instead of UIColor.
    #define GREEN_COLOR [UIColor greenColor]
    #define YELLOW_COLOR [UIColor yellowColor]
    #define BLACK_COLOR [UIColor blackColor]

    // For each path in the array of paths...
    for (id object in paths) {
        CGPathRef path = (__bridge CGPathRef)object;

        // Set the background of the path to yellow.
        CGContextSetFillColorWithColor(context, [YELLOW_COLOR
CGColor]);

        CGContextAddPath(context, path);
        CGContextFillPath(context);

        CGContextDrawPath(context, kCGPathStroke);

        // Create a frame for this path and draw the text.
        CTFrameRef frame = CTFramesetterCreateFrame(framesetter,
                                    CFRangeMake(startIndex, 0),
path, NULL);
        CTFrameDraw(frame, context);

        // Start the next frame at the first character not visible in
```

```
this frame.
        CFRange frameRange = CTFrameGetVisibleStringRange(frame);
        startIndex += frameRange.length;
        CFRelease(frame);
}

CFRelease(attrString);
CFRelease(framesetter);
}
```