

Programming with Haiku

Lesson 16

Written by DarkWorm



Fonts are one of those things which some people go ga-ga over – maintaining a collection of thousands for no apparent reason except for liking them – and others couldn't possibly care less about. Regardless of the camp to which you belong, knowing even a little bit about how to manipulate text display in Haiku is quite handy.

Using Fonts in Haiku

Like many other kinds of add-ons in Haiku, there is a folder dedicated to holding fonts the user wishes to add to the system. The `find_directory()` constant for it is `B_COMMON_FONTS_DIRECTORY`, which maps to `/boot/common/data/fonts` in Haiku and `/boot/home/config/fonts` for Zeta and BeOS R5. R5 and Zeta required fonts to be sorted into subfolders, *ttfonts* for TrueType fonts and *psfonts* for PostScript. Haiku removes this restriction. Haiku also removes the requirement to reboot the system to use newly-installed fonts – copy them in and away you go.

Since the days of BeOS R5, font handling has changed considerably. Text rendering quality was a sore spot for some BeOS users. R5 did not provide very good antialiasing for text. The situation improved considerably in Zeta with the change to a different font engine – one developed by the renowned font company Bitstream. However, this improvement bit the dust along with the rest of Zeta. Haiku, fortunately, uses the open source FreeType library to provide excellent font rendering and make more kinds of fonts usable than ever before.

A Bit About Typography



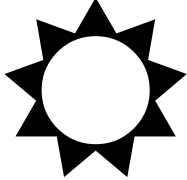
No lesson about fonts would be complete without a little background about typography, which is the process of arranging type for printing. The basics of typography were started with Gutenberg's printing press, and it has steadily grown more complicated ever since. You won't need to have an exhaustive knowledge of the subject to use text in your applications, but knowing some of terms is quite beneficial.

Fonts are grouped together into families. A font family is little more than a name. For example, Times New Roman, Century Schoolbook L, and Helvetica are all examples of well-known font families. Each family contains one or more individual styles or **faces**. While you will easily recognize Italic and Bold as being familiar, there are others, as well. Sometimes you will hear the term **font weight** when styles are discussed. A font's weight is the heaviness of the type. Here is a non-exhaustive list of font weights, from the lightest to the heaviest.

- Extra light
- Light
- Book
- Normal / Regular / Plain
- Medium
- Demi-bold
- Bold
- Black
- Extra Black

Font families are often grouped into generalized categories. The specific categories used depend on whom you ask, but for our purposes we will use three basic categories: serif, sans serif, and decorative. Serif fonts are the most common in reading literature. A **serif** is a

decorative stroke appearing in certain locations of letters. Characters in sans serif fonts do not have them. Decorative fonts are most often used for logos or specific effects. Here are examples of the capital letter R as rendered in three different fonts.

Times New Roman	Arial	Wingdings
		

Times New Roman is a serif font, Arial is sans serif, and Wingdings is decorative. Notice the little lines extending out to the sides at the ends of the legs of the R shown in Times New Roman. These are serifs.

Along with these terms, here are some others that you will see:

Leading – Pronounced to rhyme with the word 'bedding', this is the amount of space placed between lines of text.

Glyph – A character used in a font. Depending on the font used this can be a letter, a number, or something else entirely.

Kerning – The amount of horizontal space placed between individual letters. Proportional fonts – those without a fixed character width – use different amounts of space between letters for better readability.

Baseline – An imaginary line upon which letters are placed.

Ascender – Part of a lowercase character which sticks up above the rest, such as the top part of a lowercase letter d.

Descender – Part of a lowercase character which hangs down below the baseline, such as the bottom part of a lowercase letter p.

Point – Points are a unit of measurement which is not to be confused with pixels. There are 72 points in an inch.

Working with Fonts

Fonts are a potentially complex subject, but what do you do if all you want to do is print some text on the screen? Not a problem. Let's start with the simplest of examples: a BView that shows some text.

1. Create a new project in Paladin with the *Main Window with GUI and Menu* template.
2. Open App.cpp and set the application signature to "application/x-vnd.test-FontDemo1" and save it.
3. Create a new file called MainView.cpp and check the box to also create a header.
Now let's get to the real work.

MainWindow.cpp

```
#include "MainWindow.h"

#include <Application.h>
#include <Menu.h>
#include <MenuItem.h>
#include <View.h>

#include "MainView.h"

MainWindow::MainWindow(void)
    : BWindow(BRect(100,100,500,400), "Font Demo", B_TITLED_WINDOW,
              B_ASYNCHRONOUS_CONTROLS)
{
    BRect r(Bounds());
    r.bottom = 20;
    fMenuBar = new BMenuBar(r, "menubar");
    AddChild(fMenuBar);

    r = Bounds();
    r.top = 20;
    MainView *view = new MainView(r);
    AddChild(view);
}

void
MainWindow::MessageReceived(BMessage *msg)
{
    switch (msg->what)
    {
        default:
        {
            BWindow::MessageReceived(msg);
            break;
        }
    }
}

bool
MainWindow::QuitRequested(void)
{
    be_app->PostMessage(B_QUIT_REQUESTED);
    return true;
}
```

MainView.h

```
#ifndef MAINVIEW_H
#define MAINVIEW_H

#include <View.h>

class MainView : public BView
{
public:
    MainView(const BRect &frame);
    void      Draw(BRect update);
};

#endif
```

MainView.cpp

```
#include "MainView.h"

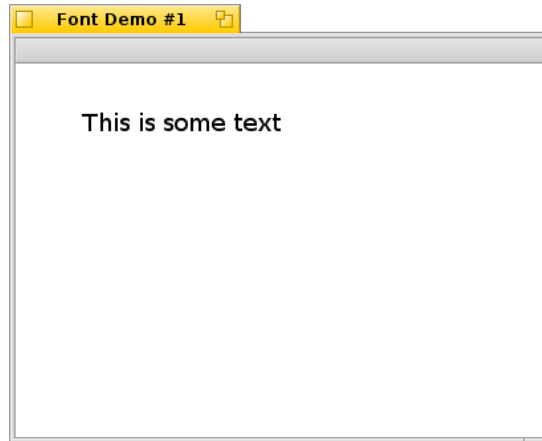
MainView::MainView(const BRect &frame)
    : BView(frame, "textview", B_FOLLOW_ALL, B_WILL_DRAW)
{
}

void
MainView::Draw(BRect update)
{
    // The Draw hook function is called whenever a BView is asked to
    // redraw itself on the screen. We will just write some code to draw
    // some text.
    BFont font;
    font.SetSize(18.0);

    SetFont(&font);

    // DrawString uses the BView's current font settings, draw mode, and
    // color to draw the text. Note that the point specified is the left
    // end of the baseline, so calculating where to draw text can be a
    // little backwards when compared to drawing anything else.
    DrawString("This is some text", BPoint(50,50));
}
```

This example is about as simple as it gets. Running the demo results in this window:



BView's `Draw()` method is not limited to merely drawing text, but we'll save that for another lesson. It is to be noted, though, that the graphics state of the `MainView` class – i.e. the pen location, font size, current high color, etc. – only changes when we change it. With that said, we could just as easily move everything except the `DrawString()` call into the constructor to make drawing faster, but this kind of optimization will only work for simpler controls.

What Do We Use?

If you take a quick peek at `Font.h`, you'll see that there are a ton of different methods for the `BFont` class. The vast majority are for specialized use, such as selecting different encodings or rotating the text. Here are the methods that you will use most:

```
void GetHeight(font_height *height) const;
```

Gets the pixel value for the font's leading, maximum ascender height, and maximum descender size, taking the current font size into account. `font_height` is just a struct containing three floats: `ascent`, `descent`, and `leading`. If you want to figure out the safe pixel height for a line of text printed with the font, call this and add the three values together.

```
void TruncateString(BString *string, uint32 mode, float maxWidth);
```

This method yanks characters from the string until it fits in `maxWidth` pixels. There are four modes: `B_TRUNCATE_BEGINNING`, `B_TRUNCATE_MIDDLE`, `B_TRUNCATE_END`, and `B_TRUNCATE_SMART`. The first three modes are obvious. The last one, according to the *Be Book*, is apparently meant to be used with the related method `GetTruncatedStrings()` to cut the strings in such a way that they are all different, paying attention to word boundaries, separators, punctuation, etc. As of R5, it was unimplemented. The Haiku implementation functions exactly the same as `B_TRUNCATE_MIDDLE`.

```
bool IsFixed(void) const;
```

Returns true if the font is a fixed-width font.

```
void SetFace(uint16 face);
uint16 Face(void) const;
```

Sets or gets the face of the font. Instead of string values, these calls rely on integer constants, which is handy if you want to specify a certain style without having to figure out its name. Here are the face constants:

- B_ITALIC_FACE
- B_UNDERSCORE_FACE
- B_NEGATIVE_FACE
- B_OUTLINED_FACE
- B_STRIKEOUT_FACE
- B_BOLD_FACE
- B_REGULAR_FACE
- B_CONDENSED_FACE*
- B_LIGHT_FACE*
- B_HEAVY_FACE*

*The CONDENSED, LIGHT, and HEAVY face constants are new in Haiku and not available to Zeta or BeOS R5 programs.

```
void SetFamilyAndFace(const font_family family, uint16 face);
void SetFamilyAndStyle(const font_family family, const font_style style);
void GetFamilyAndStyle(font_family *family, font_style *style);
```

These methods make for an easy way to get or set a specific style.

```
void SetSize(float size);
float Size(void) const;
```

Set the point size for the font. As of this writing, the Be Book states a limit of 10,000 points, but Haiku does not have this limitation, unlike BeOS.

```
float StringWidth(const char *string) const;
float StringWidth(const char *string, int32 length) const;
```

Returns the width of string in the current size.

```
int32 count_font_families(void);
int32 count_font_styles(font_family family);
status_t get_font_family(int32 index, font_family *family,
                        uint32 *flags = NULL);
status_t get_font_style(font_family family, int32 index, font_style *style,
                       uint32 *flags = NULL);
```

These global functions are used to iteratively read all fonts installed on the system.

Other Tips and Tricks

- There are three global font objects: `be_plain_font`, `be_bold_font`, and `be_fixed_font`. These built-ins are set by the user with the Fonts preferences application. Use these fonts in your applications whenever possible to help keep a consistent look to the interface.
- If you draw text on a colored background, make sure you set the view's low color to that of the background or else your text will look kinda weird.
- To draw text on top of a picture, set the view's drawing mode to `B_OP_ALPHA` before drawing the text.
- Tweaks can be made to the kerning with the methods related to escapements.

Going Further

- Play around with the `FontDemo` demonstration app bundled with Haiku. See how each of the different effects changes the text.
- Try using a `BMessageRunner` to make an animated text demo that changes sizes, rotates the text, or something else cool.

Answers to Unit 2 Review

Lesson 6

1. The `BApplication` class sets up communications with the `app_server`.

Lesson 7

1. The Support Kit is a collection of classes used to support the other kits with general-purpose classes for string handling, memory management, and more.
2. `MessageReceived()` handles messages sent to a `BWindow`, as well as any `BHandler` subclass, such as `BApplication`, `BLooper`, and `BView`.
3. `ResizeToPreferred()` causes a `BButton` to resize itself to the smallest size which properly displays the label. We should use it because it makes our code work regardless of the size of the font used in the GUI.

Lesson 8

1. All GUI controls ultimately inherit from the `BView` class.
2. `B_WILL_DRAW` tells the `app_server` that the `BView` in question needs to be sent redraw messages.
3. The difference between `Bounds()` and `Frame()` is that `Bounds()` returns the size of the `BWindow` or `BView` and `Frame()` returns its size and location in its parent's coordinates.

Lesson 9

1. `SetSelectionMode()` sets the message sent whenever a `BListView`'s selection changes. The invocation message, set by `SetInvocationMessage()`, sets the message sent when the user double-clicks on an item in a `BListView`.
2. `SetTarget()` chooses a `BLooper` or `BHandler` which is to become the target of messages sent by a control.

Lesson 10

1. The main difference between a BFile and a BEntry is that a BEntry handles a file or directory's presence in the filesystem, such as its location and other related data, whereas a BFile manipulates a file's data, such the information stored in the file and its attributes.
2. An entry_ref is a lightweight structure which can describe the location of a file or directory on disk.
3. entry_ref instances may not point to a location which does not exist, and they do not consume a file handle, unlike BEntry objects.

Lesson 11

1. Yes. It is a subclass of BNode, which has methods to manipulate attributes.
2. A BString object's Lock() method must be called to get a pointer which can be passed to BFile::Read(). Care also must be taken to ensure that data read from the BFile is not greater than the size specified in BString::Lock().

Lesson 12

1. Metadata is information about a file which is not part of the file's data, such as its modification time.
2. An indexed attribute may be no larger than 255 bytes.
3. African or European?
4. A bool may not be used in an indexed attribute. Data of type char can be because the char type is simply reinterpreted integer data.
5. BMimeType should be used to set the preferred application for an entire file type, whereas BNodeInfo should be used for individual files.

Lesson 13

1. lsindex will tell you the names of all indexed attributes on a volume.
2. mkindex can make an attribute able to be queried.
3. Having a large number of indexed attributes on a volume will slow down all queries on that volume.
4. Reverse Polish Notation is an entry system where the operator is entered after each operand instead of between them.

Bonus: The functions fs_open_index_dir(), fs_read_index_dir(), and fs_close_index_dir() are used to programmatically get the names of indexed attributes on a volume.

Lesson 14

1. B_WATCH_FILE is not a node-monitoring flag.
2. entry_ref and node_ref objects need to be stored away to handle B_ENTRY_REMOVED operations because the node-monitoring message sent does not provide the removed entry's name, making it impossible to create an entry_ref strictly based on the information provided in the message. It does, however, provide enough to construct a node_ref.

Lesson 15

1. 0.5 [0:32] ("FOOBAR\000\000")