# Surpass Manual

Ana Nelson

March 22, 2009

# Contents

To compile this documentation requires LaTeX, Idiopidae, and my branch of webby, which is available from a bzr repository at http://ananelson.com/code/webby/.

This documentation refers to Surpass version 0.0.5.

# Chapter 1

# Installation and Hello World

## 1.1 Dependencies

Surpass only needs basic Ruby. It has been tested using Ruby 1.8.6 and JRuby 1.1.5.

For development, you will want to have access to something that can open Microsoft Excel files. This could be Microsoft Excel, Open Office, Google Docs or even a gmail account.

## 1.2 Gem Installation

```
sudo gem install surpass
```

## 1.3 Source Installation

```
bzr branch http://ananelson.com/code/surpass
cd surpass
sudo rake gem:install
```

## 1.4 Hello World

Let's do a minimal "Hello World" script. We'll need to take care of any imports, initialize a Workbook object, create a Worksheet within the workbook, then write some text. Here's how.

### 1.4.1 Surpass

```ruby
require 'rubygems'
require 'surpass'

book = Workbook.new
sheet = book.add_sheet

sheet.write(0, 0, "Hello World!")

book.save("content/examples/hello-world.xls")
```

### 1.4.2 Result

And, here's how it looks.
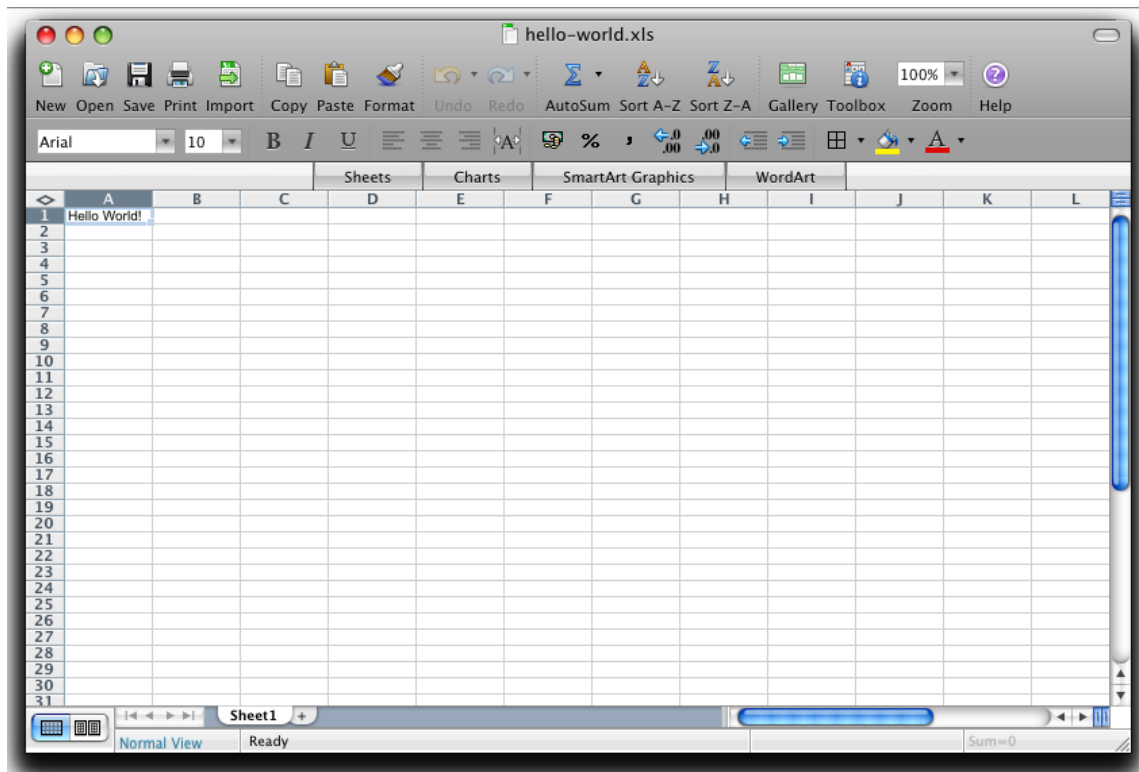
# Chapter 2

# Writing Data

## 2.1   write

The basic method for writing data to cells is the worksheet's write method.

```ruby
273    def write(r, c, label = "", style = nil)
274      row(r).write(c, label, style)
275    end
276
```

There are two required arguments, the row and column. These are zero-based indexes for the row and column. To write to the first cell in the spreadsheet, you would pass 0, 0. The next argument is the label, this is the value you want written in the cell. This defaults to an empty string (for no particular reason). You can write nil, a String, a Boolean, a Numeric or a Date format. If you pass an object belonging to an unsupported class, you will get an error message, and in this case you should call some method on your object which will return a String or whatever the appropriate label is. The fourth argument is for style, which should be either nil, true, a hash or an instance of the StyleFormat class. See the chapter on Formatting for more information. The style parameter defaults to nil, which means that the default Excel format will be applied.

## 2.2   Writing Arrays of Data

Frequently, you may want to write more than one value at a time, and so Surpass has convenience methods which handle arrays for you. In the background, these are just looping over the array and calling write() for each value you pass. There's no magic here and, for now at least, no clever optimization. The available methods are write_array_to_row, write_array_to_column, and write_arrays. The write_arrays method expects an array of arrays, the first two expect a single array.

```ruby
277    def write_array_to_row(array, r, c = 0, style = true)
278      array.each_with_index do |a, i|
279        row(r).write(c + i, a, style)
280      end
281    end
282
283    def write_array_to_column(array, c, r = 0, style = true)
284      array.each_with_index do |a, i|
285        row(r + i).write(c, a, style)
286      end
287    end
288
289    def write_arrays(r, c, array_of_arrays, style = true)
290      array_of_arrays.each_with_index do |a, i|
```

```
291        raise "not an array of arrays!" unless a.is_a?(Array)
292        write_array_to_row(a, r + i, c, style)
293      end
294    end
295
```

## 2.3 Autoformatting

Autoformats are number formats which are automatically applied to Dates, Floats and similar classes. To have autoformats applied, then pass true as the style parameter to the write function.

Here is the relevant code from row.rb:

```
111      when TrueClass # Automatically apply a nice numeric format.
112        case label
113        when DateTime, Time
114          style = @parent_wb.styles.default_datetime_style
115        when Date
116          style = @parent_wb.styles.default_date_style
117        when Float
118          style = @parent_wb.styles.default_float_style
119        else
120          style = @parent_wb.styles.default_style
121        end
122
```

And here are the default formats being defined in style.rb:

```
100    def default_date_style
101      @default_date_style ||= StyleFormat.new(:number_format_string => 'dd-mmm-yyyy')
102    end
103
104    def default_datetime_style
105      @default_datetime_style ||= StyleFormat.new(:number_format_string => 'dd-mmm-yyyy hh:mm:ss')
106    end
107
108    def default_float_style
109      @default_float_style ||= StyleFormat.new(:number_format_string => '#,##0.00')
110    end
111
```

If you use any of the array-writing methods, then autoformatting will be applied by default. To override this behaviour you can pass your own StyleFormat or nil to use the generic default format.

```
1    require 'rubygems'
2    require 'surpass'
3
4    book = Workbook.new(__FILE__.gsub(/rb$/, "xls"))
5    sheet = book.add_sheet
6
7    # Passing true for the style parameter to write will invoke autoformatting.
8    sheet.write(0, 0, "Hello World!", true)
9    sheet.write(1, 0, 1, true)
10   sheet.write(2, 0, 1.0, true)
11   sheet.write(3, 0, Date.today, true)
12   sheet.write(4, 0, DateTime.now, true)
13   sheet.write(5, 0, Time.now, true)
14
```

```
15   array_of_arrays = [
16     [1, 2, 3],
17     [1.0, 2.0, 3.0],
18     [Date.today, DateTime.now],
19     %w{a b c}
20   ]
21
22   # Writing arrays will automatically autoformat.
23   sheet.write(7, 0, "With autoformat:")
24   sheet.write_arrays(8, 0, array_of_arrays)
25
26   # Unless you specify your own format, or nil for a generic default.
27   sheet.write(13, 0, "Without autoformat:")
28   sheet.write_arrays(14, 0, array_of_arrays, nil)
29
30   sheet.set_column_widths(0..2, 20)
31
32   book.save
33
```
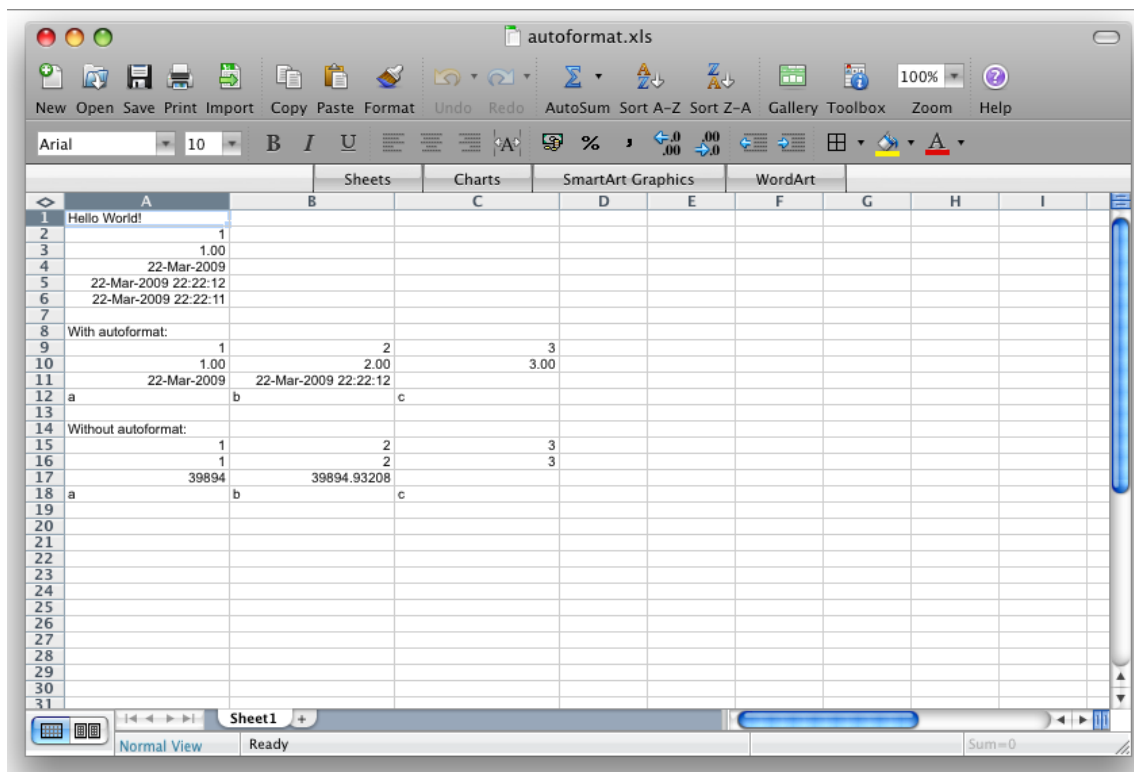
# Chapter 3

# Formatting

## 3.1 Reference

There is a command line tool included with Surpass which provides some useful reference data:

```
surpass -h


Usage: surpass-info [options]
    -c, --colors, --colours        List available colors
    -p, --patterns                 List available patterns
    -h, --help                     Show this message
```

And since you are running this on the command line, you can save or pipe the output to other commands:

```
surpass -c | grep green


bright-green
dark-green
green
light-green
olive-green
sea-green
```

## 3.2 Formatting

The StyleFormat class is a wrapper for the various types of formatting you can apply to a cell. StyleFormat has attributes:

- number_format_string

- font

- alignment
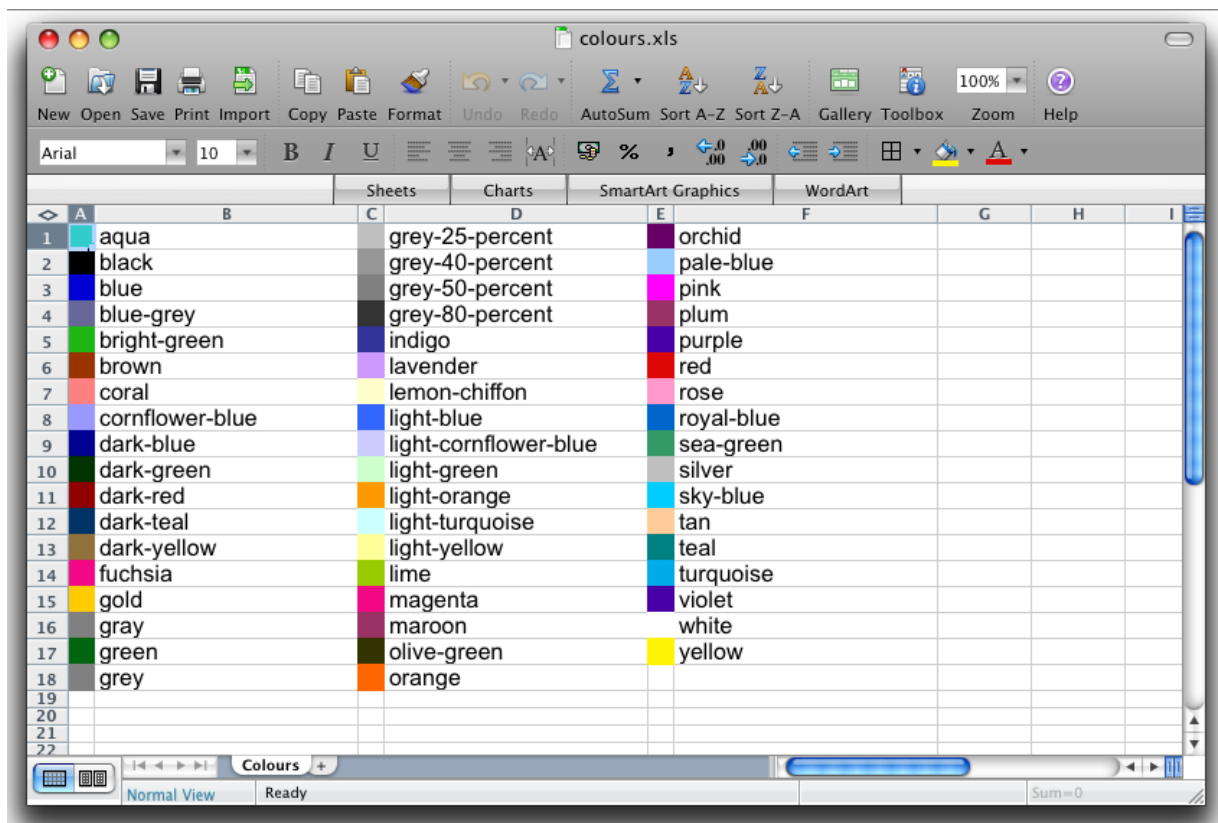
- borders

- pattern

- protection

Each of these attributes has a corresponding class, and you can look in lib/formatting.rb for the source.

There are two basic ways to set formatting options. You can pass a hash with formatting options when you initialize a new StyleFormat instance, or you can set individual attributes of the formatting classes. You can combine both approaches. Both of these are demonstrated in the examples in this section.

### 3.2.1   Specifying Colours
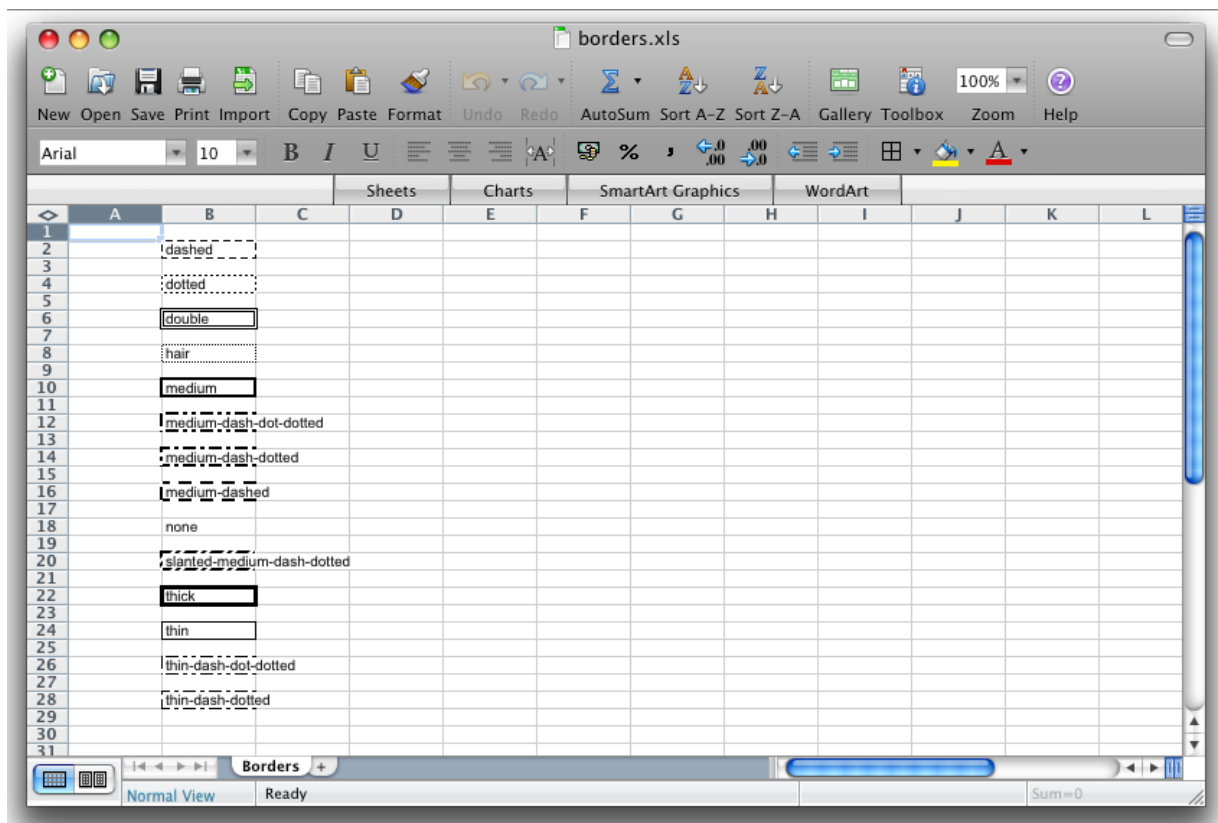
Here is a list of available colours:

```
aqua                    grey-25-percent         orchid
black                   grey-40-percent         pale-blue
blue                    grey-50-percent         pink
blue-grey               grey-80-percent         plum
bright-green            indigo                  purple
brown                   lavender                red
coral                   lemon-chiffon           rose
cornflower-blue         light-blue              royal-blue
dark-blue               light-cornflower-blue   sea-green
dark-green              light-green             silver
dark-red                light-orange            sky-blue
dark-teal               light-turquoise         tan
dark-yellow             light-yellow            teal
fuchsia                 lime                    turquoise
gold                    magenta                 violet
gray                    maroon                  white
green                   olive-green             yellow
grey                    orange
```



12

### 3.2.2 Border Formats

Here is a list of available border line types:

```
none
thin
medium
dashed
dotted
thick
double
hair
medium-dashed
thin-dash-dotted
medium-dash-dotted
thin-dash-dot-dotted
medium-dash-dot-dotted
slanted-medium-dash-dotted
```
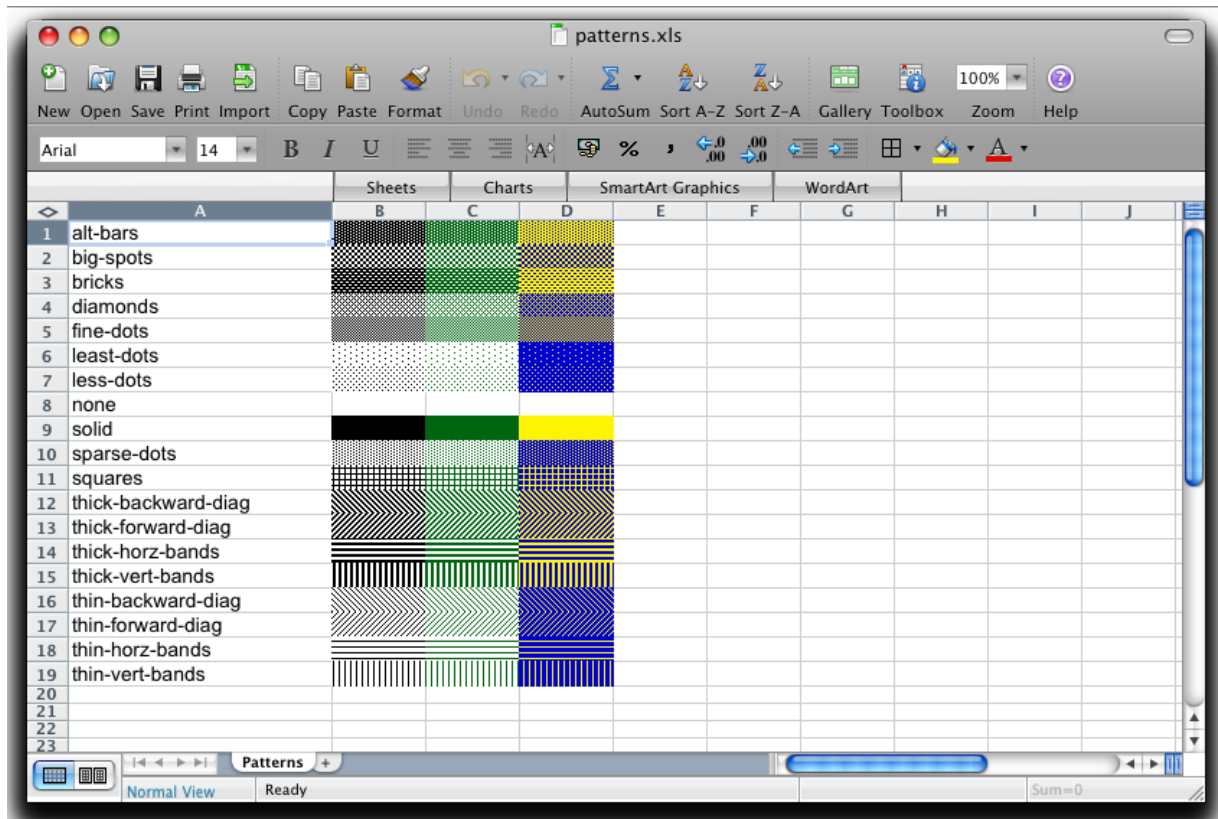


### 3.2.3 Fill Patterns

Here is a list of available fill patterns:

```
none
solid
fine-dots
alt-bars
sparse-dots
thick-horz-bands
thick-vert-bands
thick-backward-diag
```

```
thick-forward-diag
big-spots
bricks
thin-horz-bands
thin-vert-bands
thin-backward-diag
thin-forward-diag
squares
diamonds
less-dots
least-dots
```



### 3.2.4  Surpass

```ruby
require 'rubygems'
require 'surpass'

book = Workbook.new(__FILE__.gsub(/rb$/, "xls"))
sheet = book.add_sheet("Demo Worksheet") # You can name your worksheets.

# Let's set up some formatting.

# Remember to use Excel-style formatting directives, not sprintf.
date_format = StyleFormat.new(:number_format_string => "DDDD DD MMM YYYY")

fancy_format = StyleFormat.new(
  :font_name => 'Times New Roman',
  :font_colour => 'green',
  :font_italic => true
)
```

```ruby
17
18    sheet.write(0, 0, "Hello World!", fancy_format)
19    sheet.write(0, 1, Date.today, date_format)
20
21    # You can also set up formatting by passing attributes directly to the constituents of StyleFormat
22
23    # Font colours.
24    Formatting::COLOURS.keys.each_with_index do |c, i|
25      format = StyleFormat.new
26      format.font.name = 'Verdana'
27      format.font.color = c
28      format.font.size = i + 5
29      sheet.write(i, 5, c, format)
30    end
31
32    # Font underlining.
33
34    [:none, :single, :single_accounting, :double, :double_accounting, nil, true, false].each_with_index
35      format = StyleFormat.new
36      format.font.underline = u
37      sheet.write(i, 7, u.to_s, format)
38    end
39
40    # Font bold, italic, strikethrough, outline are simple booleans.
41    [:bold, :italic, :struck_out, :outline].each_with_index do |s, i|
42      attribute = ("font_" + s.to_s).to_sym
43      sheet.write(i, 8, s.to_s, StyleFormat.new(attribute => true))
44    end
45
46    # Cell alignment.
47    sheet.write(15, 2, "top left", :text_align => 'top left',
48      :border_top => 'pink',
49      :border_left => 'pink'
50    )
51    sheet.write(15, 3, "top center", :text_align => 'top center')
52    sheet.write(15, 4, "top right", :text_align => 'top right')
53    sheet.write(16, 2, "bottom left", :text_align => 'bottom left')
54    sheet.write(16, 3, "bottom centre", :text_align => 'bottom centre')
55    sheet.write(16, 4, "bottom right", :text_align => 'bottom right',
56      :border_bottom => 'pink',
57      :border_right => 'pink'
58    )
59
60
61    # Borders
62    sheet.write(3, 1, "borders",
63      :border_right => 'medium blue',
64      :border_left => 'yellow', # thin by default
65      :border_top => 'dotted purple',
66      :border_bottom => 'dashed' # black by default
67    )
68
69    # Or the hash-free option.
70    crazy_border_format = StyleFormat.new
71    crazy_border_format.borders.all = 'slanted-medium-dash-dotted grey'
72    crazy_border_format.pattern.fill = 'light-cornflower-blue'
```
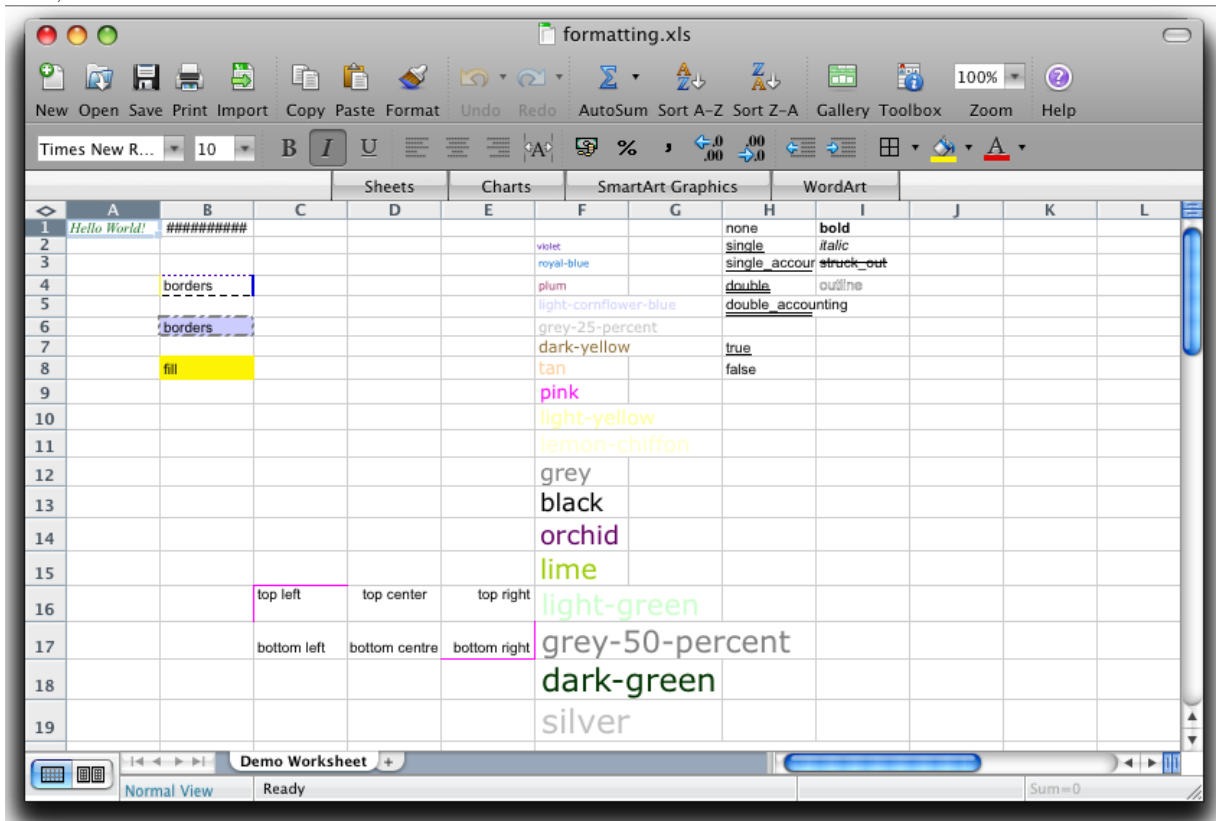
```
73
74  sheet.write(5, 1, "borders", crazy_border_format)
75
76  sheet.write(7, 1, "fill", :fill_color => 'yellow')
77
78  book.save
79
```

And, here's how it looks.

# Chapter 4

# Saving

Typically, you will call the workbook's save() method to write that workbook to a file. You can pass the filename as an argument to save(), or as an argument to new() when you first instantiate a workbook object.

However, you can also call a workbook's data() method, which gives you direct access to a workbook's binary data. You can write this to a file manually, as in this example:

```ruby
require 'rubygems'
require 'surpass'

book = Workbook.new
sheet = book.add_sheet

sheet.write(0, 0, "Hello World!")

File.open(__FILE__.gsub(/rb$/, "xls"), "w") do |f|
  f.write book.data
end

```

Or, you could use this data as an argument to Rails' send_data method.