

Surpass Manual

Ana Nelson

July 2, 2010

Contents

1	Installation and Hello World	5
1.1	Dependencies	5
1.2	Gem Installation	5
1.3	Source Installation	5
1.4	Hello World	5
1.4.1	Surpass	5
1.4.2	Result	5
2	Writing Data	7
2.1	write	7
2.2	Writing Arrays of Data	7
2.3	Autoformatting	8
3	Formatting	11
3.1	Reference	11
3.2	StyleFormat Class	11
3.3	Number Format Strings	11
3.3.1	Specifying Colours	12
3.3.2	Border Formats	13
3.3.3	Fill Patterns	14
3.3.4	Surpass	15
4	Saving	19
5	Formulas	21
	To compile this documentation requires \LaTeX and the gems Gorgyrella and Webby.	
	This documentation refers to Surpass version 0.1.0.	

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

```
require 'rubygems'
require 'surpass'

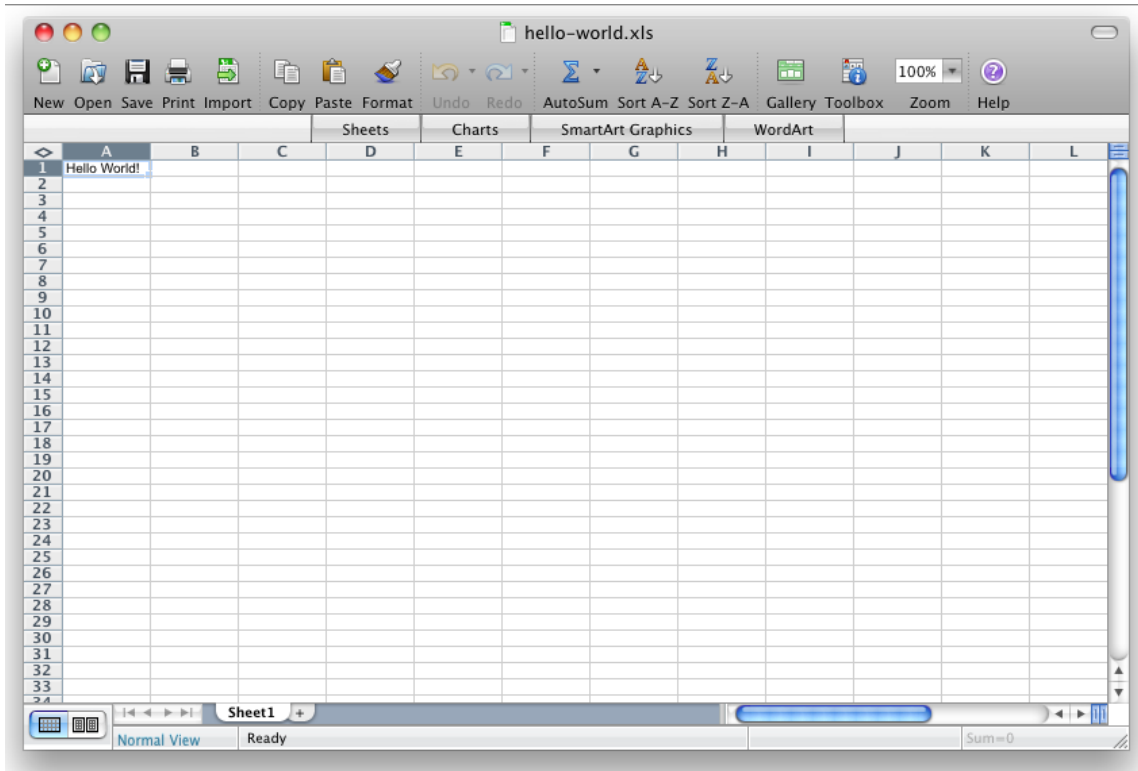
book = Workbook.new
sheet = book.add_sheet

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.

```
def write(r, c, label = "", style = nil)
  if label.is_a?(Array)
    if label[0].is_a?(Array)
      write_arrays(r, c, label, style || true)
    else
      write_array_to_row(label, r, c, style || true)
    end
  else
    row(r).write(c, label, style)
  end
end
```

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.

```
def write_array_to_row(array, r, c = 0, style = true)
  array.each_with_index do |a, i|
    row(r).write(c + i, a, style)
  end
end

def write_array_to_column(array, c, r = 0, style = true)
```

```

    array.each_with_index do |a, i|
      row(r + i).write(c, a, style)
    end
  end

  def write_arrays(r, c, array_of_arrays, style = true)
    array_of_arrays.each_with_index do |a, i|
      raise "not an array of arrays!" unless a.is_a?(Array)
      write_array_to_row(a, r + i, c, style)
    end
  end
end

```

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:

```

when TrueClass # Automatically apply a nice numeric format.
  case label
  when DateTime, Time
    style = @parent_wb.styles.default_datetime_style
  when Date
    style = @parent_wb.styles.default_date_style
  when Float
    style = @parent_wb.styles.default_float_style
  else
    style = @parent_wb.styles.default_style
  end
end

```

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

```

def default_date_style
  @default_date_style ||= StyleFormat.new(:number_format_string => 'dd-mmm-yyyy')
end

def default_datetime_style
  @default_datetime_style ||= StyleFormat.new(:number_format_string => 'dd-mmm-yyyy hh:mm:ss')
end

def default_float_style
  @default_float_style ||= StyleFormat.new(:number_format_string => '#,##0.00')
end

```

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.

```

require 'rubygems'
require 'surpass'

book = Workbook.new(__FILE__.gsub(/rb$/, "xls"))
sheet = book.add_sheet

row = -1

sheet.write(row += 1, 0, "surpass #{Surpass::VERSION} running on #{RUBY_DESCRIPTION}")

```

```

# Passing true for the style parameter to write will invoke autoformatting.
sheet.write(row+=2, 0, "Hello World!", true)
sheet.write(row+=1, 0, 1, true)
sheet.write(row+=1, 0, 1.0, true)
sheet.write(row+=1, 0, Date.today, true)
sheet.write(row+=1, 0, DateTime.now, true)
sheet.write(row+=1, 0, Time.now, true)

array_of_arrays = [
  [1, 2, 3],
  [1.0, 2.0, 3.0],
  [Date.today, DateTime.now],
  %w{a b c}
]

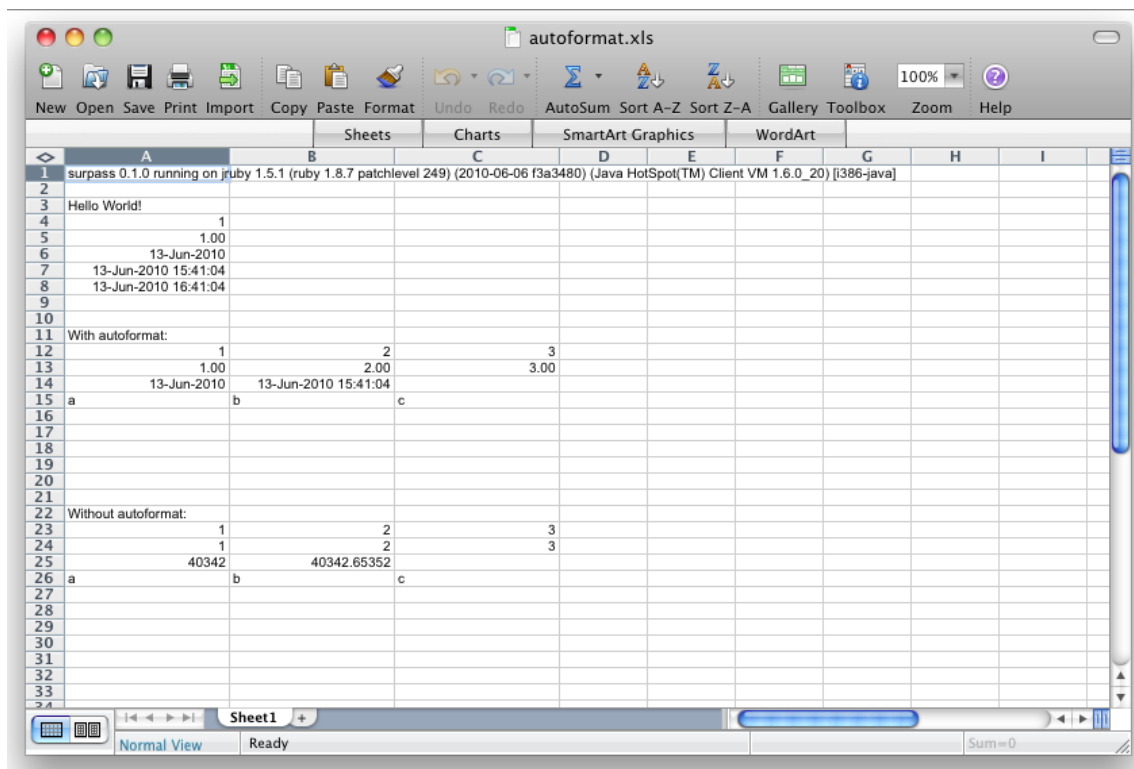
# Writing arrays will automatically autoformat...
sheet.write(row+=3, 0, "With autoformat:")
sheet.write_arrays(row+=1, 0, array_of_arrays)

# ...unless you specify your own format, or nil for a generic default.
sheet.write(row+=10, 0, "Without autoformat:")
sheet.write_arrays(row+=1, 0, array_of_arrays, nil)

sheet.set_column_widths(0..2, 20)

book.save

```



Chapter 3

Formatting

3.1 Reference

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

```
surpass -h
```

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

```
surpass -c | grep green
```

3.2 StyleFormat Class

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 (except for number_format_string) has a corresponding class, and you can look in lib/surpass/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 also combine both approaches, initializing with a hash and then modifying attributes.

3.3 Number Format Strings

This attribute is a simple string, specifying the numeric/date format to be applied to the value stored in a cell.

```

require 'rubygems'
require 'surpass'

book = Workbook.new(__FILE__.gsub(/rb$/, "xls"))
sheet = book.add_sheet

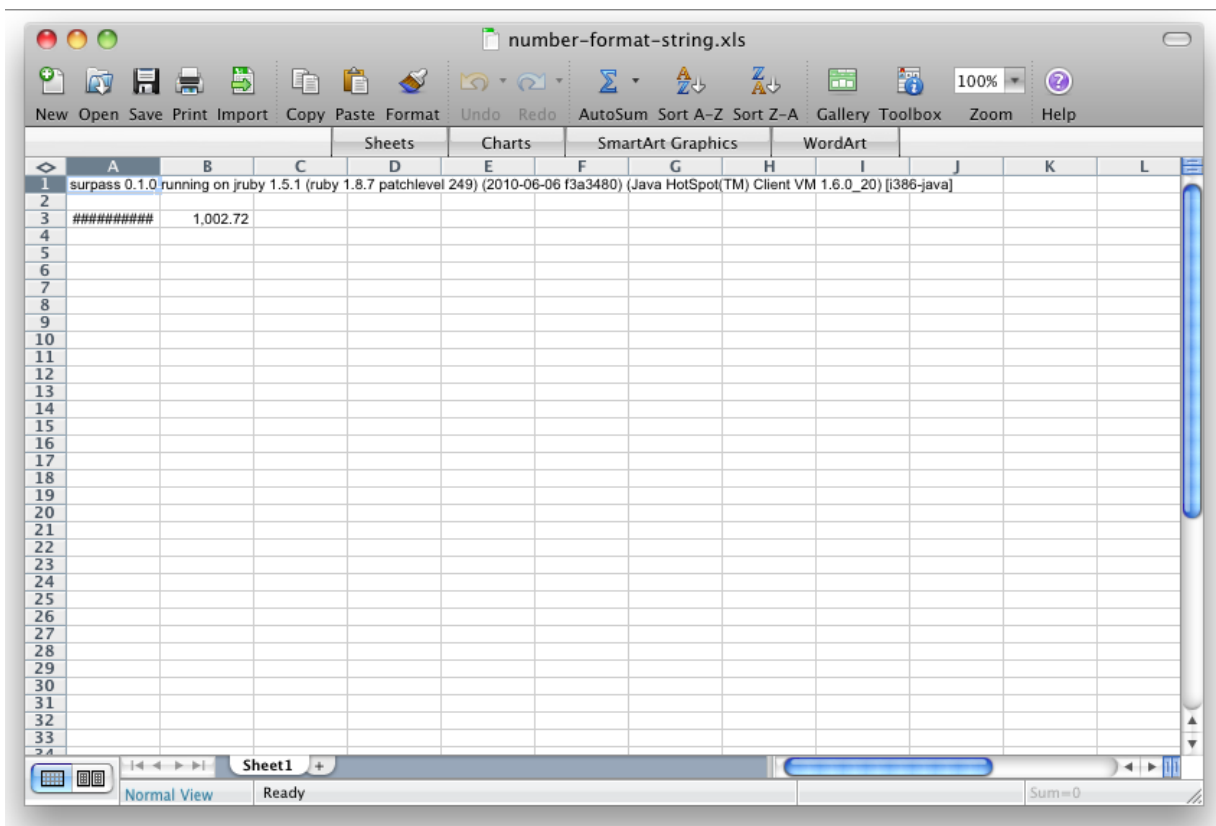
sheet.write(0, 0, "surpass #{Surpass::VERSION} running on #{RUBY_DESCRIPTION}")

date_format = StyleFormat.new(:number_format_string => 'DDD MMM YYYY')
sheet.write(2, 0, Date.today, date_format)

two_dp_format = StyleFormat.new
two_dp_format.number_format_string = "#,##0.00"
sheet.write(2, 1, 1002.71828, two_dp_format)

book.save

```

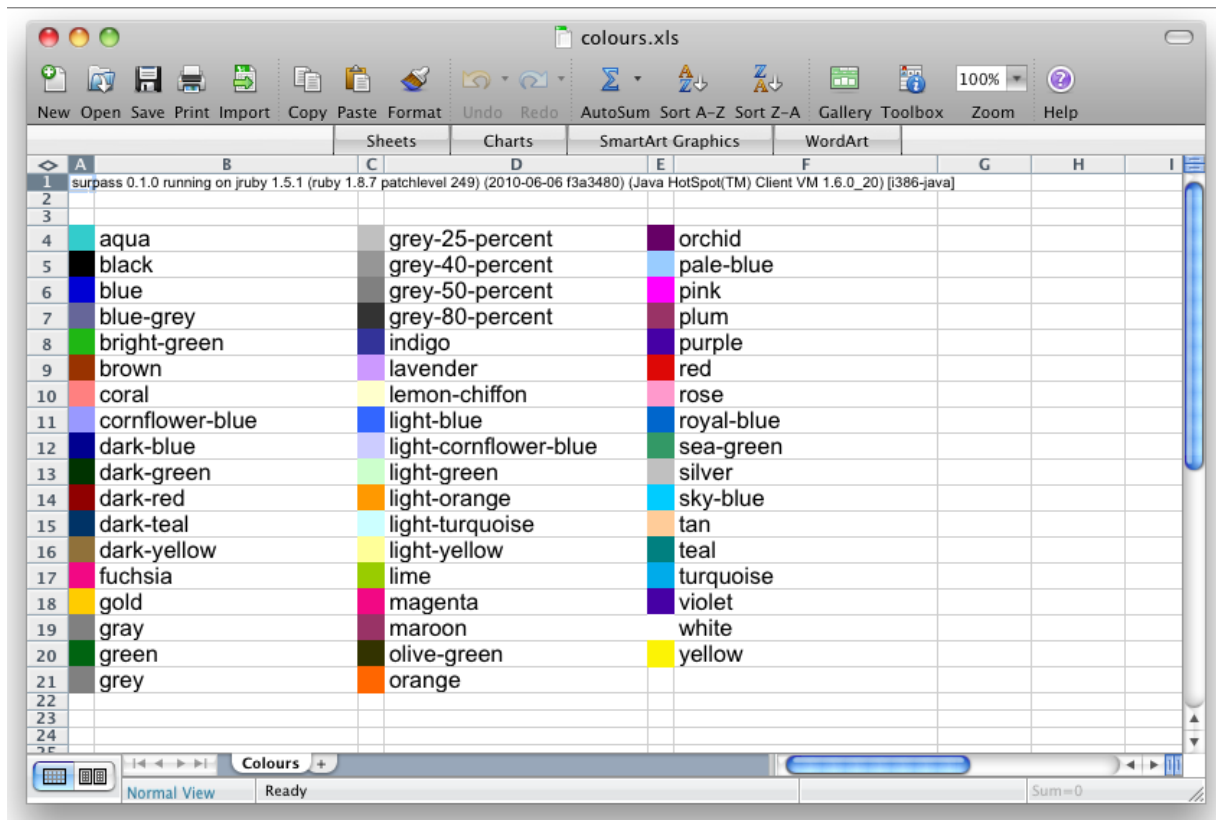


3.3.1 Specifying Colours

Here is a list of available colours:

aqua	dark-blue	green
black	dark-green	grey
blue	dark-red	grey-25-percent
blue-grey	dark-teal	grey-40-percent
bright-green	dark-yellow	grey-50-percent
brown	fuchsia	grey-80-percent
coral	gold	indigo
cornflower-blue	gray	lavender

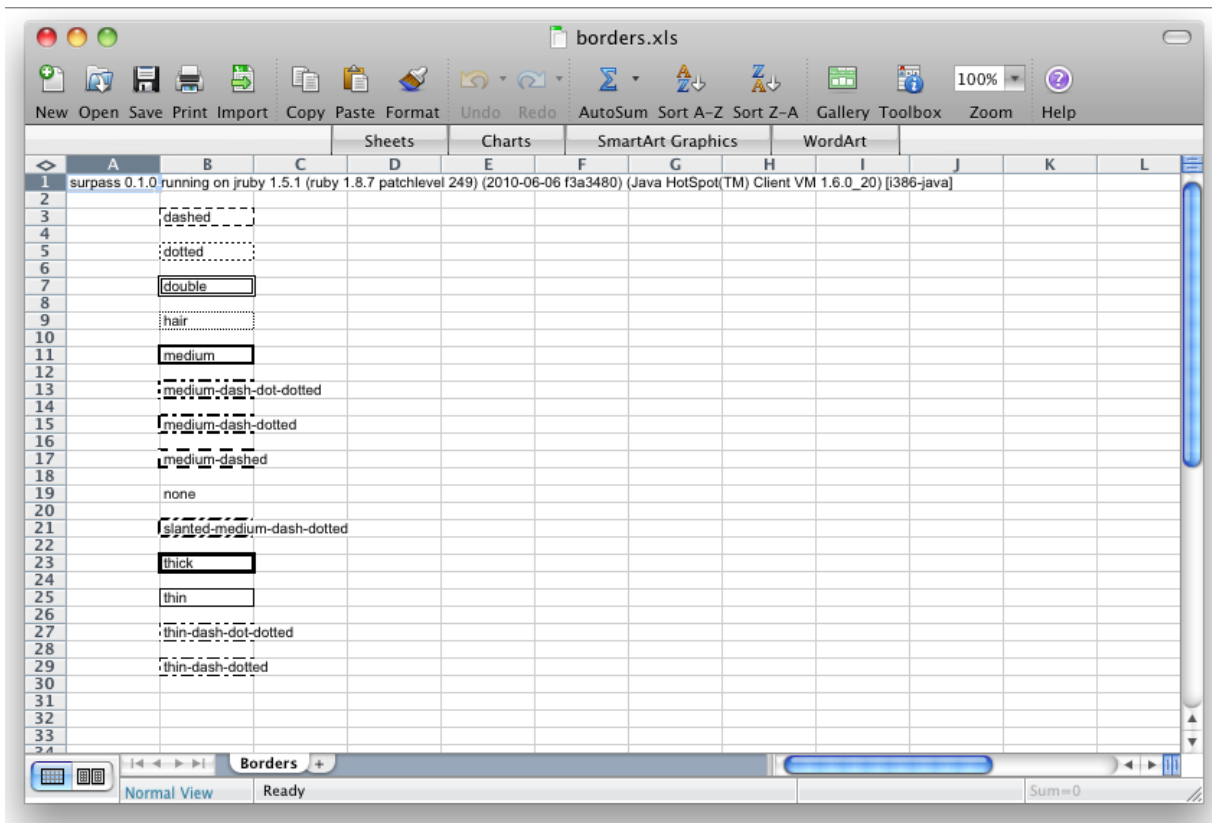
lemon-chiffon	olive-green	sea-green
light-blue	orange	silver
light-cornflower-blue	orchid	sky-blue
light-green	pale-blue	tan
light-orange	pink	teal
light-turquoise	plum	turquoise
light-yellow	purple	violet
lime	red	white
magenta	rose	yellow
maroon	royal-blue	



3.3.2 Border Formats

Here is a list of available border line types:

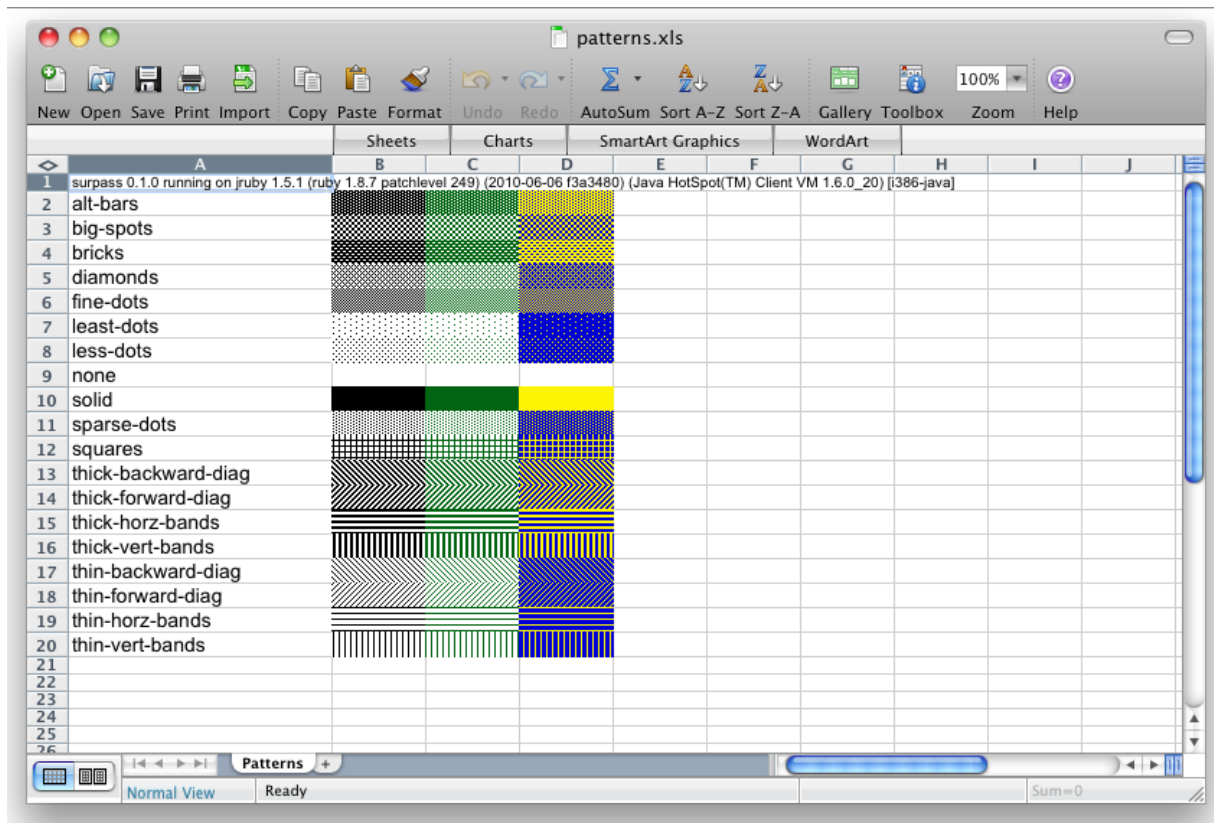
- none
- thin
- medium
- dashed
- dotted
- thick
- double
- hair
- medium-dashed
- thin-dash-dotted
- medium-dash-dotted
- thin-dash-dot-dotted
- medium-dash-dot-dotted
- slanted-medium-dash-dotted



3.3.3 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



3.3.4 Surpass

```
require 'rubygems'
require 'surpass'

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

sheet.write(0, 0, "surpass #{Surpass::VERSION} running on #{RUBY_DESCRIPTION}")

# 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
)

sheet.write(2, 0, "Hello World!", fancy_format)
sheet.write(2, 1, Date.today, date_format)

# You can also set up formatting by passing attributes directly to the constituents of StyleFormat

# Font colours.
Formatting::COLOURS.keys.each_with_index do |c, i|
  format = StyleFormat.new
```

```

    format.font.name = 'Verdana'
    format.font.color = c
    format.font.size = i + 5
    sheet.write(i+2, 5, c, format)
end

# Font underlining.

[:none, :single, :single_accounting, :double, :double_accounting, nil, true, false].each_with_index
  format = StyleFormat.new
  format.font.underline = u
  sheet.write(i+2, 7, u.to_s, format)
end

# Font bold, italic, strikethrough, outline are simple booleans.
[:bold, :italic, :struck_out, :outline].each_with_index do |s, i|
  attribute = ("font_" + s.to_s).to_sym
  sheet.write(i+2, 8, s.to_s, StyleFormat.new(attribute => true))
end

# Cell alignment.
sheet.write(15, 2, "top left", :text_align => 'top left',
  :border_top => 'pink',
  :border_left => 'pink'
)
sheet.write(15, 3, "top center", :text_align => 'top center')
sheet.write(15, 4, "top right", :text_align => 'top right')
sheet.write(16, 2, "bottom left", :text_align => 'bottom left')
sheet.write(16, 3, "bottom centre", :text_align => 'bottom centre')
sheet.write(16, 4, "bottom right", :text_align => 'bottom right',
  :border_bottom => 'pink',
  :border_right => 'pink'
)

# Borders
sheet.write(3, 1, "borders",
  :border_right => 'medium blue',
  :border_left => 'yellow', # thin by default
  :border_top => 'dotted purple',
  :border_bottom => 'dashed' # black by default
)

# Or the hash-free option.
crazy_border_format = StyleFormat.new
crazy_border_format.borders.all = 'slanted-medium-dash-dotted grey'
crazy_border_format.pattern.fill = 'light-cornflower-blue'

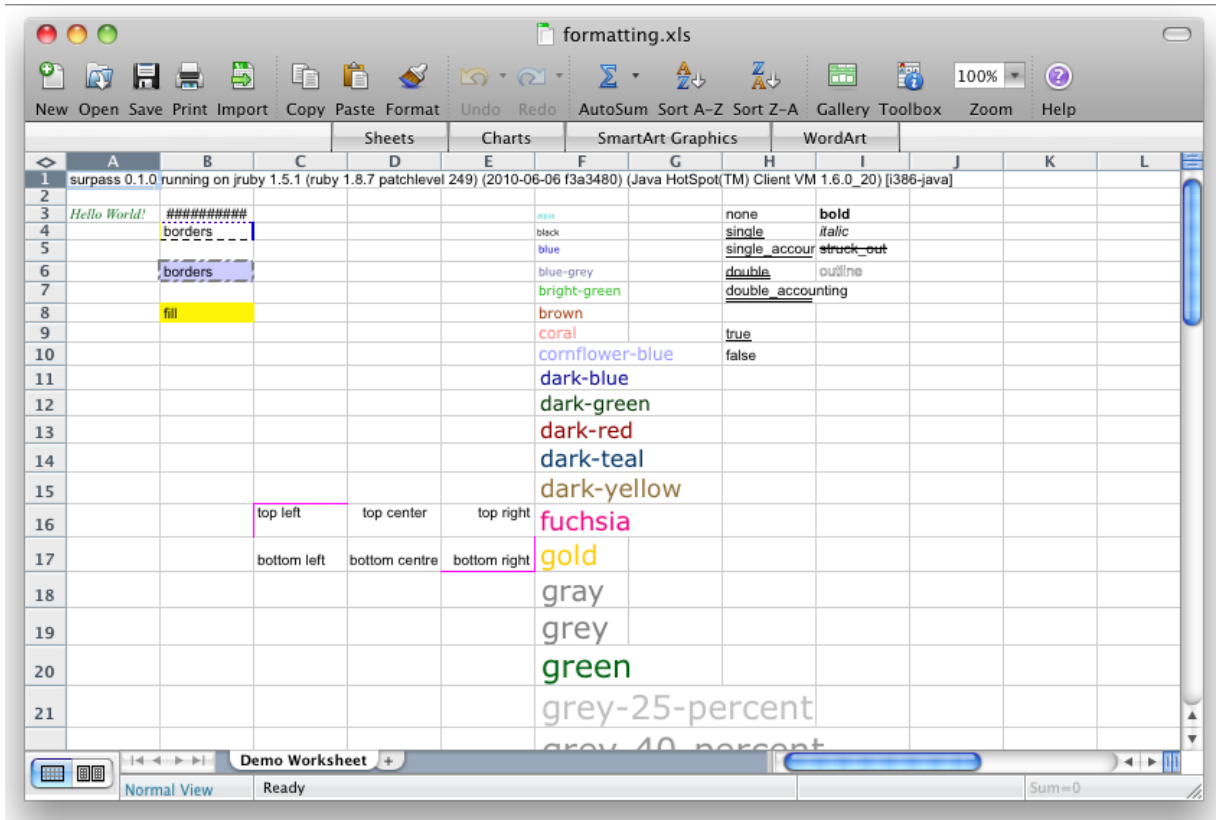
sheet.write(5, 1, "borders", crazy_border_format)

sheet.write(7, 1, "fill", :fill_color => 'yellow')

book.save

```

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:

```
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.

Chapter 5

Formulas

```
require 'rubygems'
require 'surpass'

book = Workbook.new
sheet = book.add_sheet

sheet.write(0, 0, "surpass #{Surpass::VERSION} running on #{RUBY_DESCRIPTION}")

sheet.write(2, 0, Formula.new("1+1"))
sheet.write(3, 0, Formula.new("-1"))
sheet.write(4, 0, Formula.new("-(1+1)"))
sheet.write(5, 0, Formula.new("-(1+1)/(-2-2)"))

sheet.write(2, 2, Formula.new("A3*B3"))
sheet.write(3, 2, Formula.new("A3*B4"))
sheet.write(4, 2, Formula.new("A6*B6"))

sheet.write(3, 3, Formula.new("5%"))

sheet.write(3, 5, Formula.new("A4*B4*sin(pi()/4)"))
sheet.write(4, 5, Formula.new("A5%*B5*pi()/1000"))

sheet.write(9, 9, Formula.new("A3>=B3"))

sheet.write(5, 2, Formula.new("C1+C2+C3+C4+C5/(C1+C2+C3+C4/(C1+C2+C3+C4/(C1+C2+C3+C4)+C5)+C5)-20.3e-"))
sheet.write(5, 3, Formula.new("C1^2"))
sheet.write(6, 2, Formula.new("SUM(C1;C2;;;;;C3;;;C4)"))
sheet.write(6, 3, Formula.new("SUM($A$1:$C$5)"))

sheet.write(7, 0, Formula.new("'lkjljllkllkl'"))
sheet.write(7, 1, Formula.new("'yuyiyiyiyi'"))
sheet.write(7, 2, Formula.new("'A8&B8&A8'"))
sheet.write(7, 3, Formula.new("'A8      & B8 & A8'"))

sheet.write(10, 2, Formula.new('TRUE'))
sheet.write(11, 2, Formula.new('FALSE'))
sheet.write(12, 4, Formula.new('IF(A1>A2;3;"hkjhjkhk"')) # Semicolon param delim.
sheet.write(12, 5, Formula.new('IF(A1<A2 , 3 , "hkjhjkhk"')) # Try using commas + adding whitespace
sheet.write(12, 7, Formula.new('CHOOSE(1, "a", "b")'))
sheet.write(12, 8, Formula.new('CHOOSE(2, "a", "b")'))
sheet.write(12, 9, Formula.new("CHOOSE(15, \"#{('a'..'z').to_a.join(\"\\\", \\\"\")}\")"))
```

Formulas

```
sheet.write(14, 0, Formula.new("pi()"))
sheet.write(14, 1, Formula.new("2*pi()"))
sheet.write(14, 2, Formula.new("sin(0)"))
sheet.write(14, 3, Formula.new('left("abcde", 2)'))
sheet.write(14, 4, Formula.new('hyperlink("http://google.com", "google")'))
sheet.write(14, 5, Formula.new("sin(pi())"))
sheet.write(14, 6, Formula.new('now()'))

book.save(__FILE__.gsub(/rb$/, "xls"))
```

