Mastering CSV in Ruby - sample chapter

More advanced CSV manipulation

The usage examples I presented in the last chapters were quite simple, so it's time to learn something more advanced and less commonly presented.

Preprocessing the data

Let's assume that we have the following file named `users.csv` with the following contents:

```
first_name,last_name,age
John,Doe,19
Tim,Doe,25
```

Now, let's load the contents into our code:

```
csv = CSV.parse(File.read('./users.csv'), headers: :first_row, return_headers: false)
csv.first['age']
# => "19"
```

We get the age value as a string instead of the integer. To get the integer we have to modify our code and use converter:

```
csv = CSV.parse(File.read('./users.csv'), headers: :first_row, return_headers: false, converters: [CSV::Converters[:integer]])
csv.first['age']
# => 19
```

Everything works as expected. By default, we have the following converters available:

- `:integer`
- `:float`

- `:numeric`
- `:date`
- `:date_time`
- `:all`

You can always check the complete list by calling `CSV::Converters.keys`

Creating own preprocessor (converter)

Writing a new preprocessor is easy as the converter is just a lambda expression. To confirm this, you can select one of the converters and invoke the call method with an argument on it:

```
CSV::Converters[:float].call("13.1")
# => 13.1
```

Let's play with the code a little bit and create an Email value object which parses the email addresses and provides two methods: domain and username:

```
class Email
  def initialize(value)
    @value = value
  end

  def to_s
    @value
  end

 def domain
   @value.split('@').last
 end

 def username
   @value.split('@').first
 end
end
```

We can now make an email converter that would replace any email address with the `Email` object:

```
csv = "first_name,email\nJohn,john@doe.com\nTim,tim@doe.com"
CSV::Converters[:email] = ->(value) { value.include?('@') ? Email.new(value) : value }
parsed_csv = CSV.parse(csv, headers: :first_row, converters: [:email])
parsed_csv.first['email'].domain
# => doe.com
```

There are two essential rules when it comes to creating converters:

- ensure that you will always return a value from the converter
- declare one argument if you want to parse the only value, declare two arguments if you would like to parse additional information about the given value

I mentioned the second argument - additional information about the given value. Let's take a closer look at it:

```
csv = "first_name,email\nJohn,john@doe.com\nTim,tim@doe.com"
conv = ->(arg1, arg2) { [arg1, arg2] }
parsed_csv = CSV.parse(csv, headers: :first_row, converters: [conv])
parsed_csv.first['email']
# => ["john@doe.com", #<struct CSV::FieldInfo index=1, line=2, header="email">]
```

Now we have access to the struct that contains the index, line, and header of the given field. It provides us with a lot of flexibility.