**Listing 6.7.** Validating the presence of a **name** attribute.   
**app/models/user.rb**

**class** **User** < ActiveRecord::Base

attr\_accessible :name, :email

validates :name, :presence => true

**end**

As discussed briefly in [Section 2.3.2](http://ruby.railstutorial.org/chapters/a-demo-app#sec:putting_the_micro_in_microposts), the use of **validates** is characteristic of Rails 3. (In Rails 2.3, we would write **validates\_presence\_of :name** instead.)

[Listing 6.7](http://ruby.railstutorial.org/chapters/modeling-and-viewing-users-one#code:validates_presence_of_name) may look like magic, but **validates** is just a method, as indeed is**attr\_accessible**. An equivalent formulation of [Listing 6.7](http://ruby.railstutorial.org/chapters/modeling-and-viewing-users-one#code:validates_presence_of_name) using parentheses is as follows:

**class** **User** < ActiveRecord::Base

attr\_accessible(:name, :email)

validates(:name, :presence => true)

**end**

Let’s drop into the console to see the effects of adding a validation to our User model:[15](http://ruby.railstutorial.org/chapters/modeling-and-viewing-users-one#fn:6.15)

$ rails console --sandbox

**>>** user = User.new(:name => "", :email => "mhartl@example.com")

**>>** user.save

=> false

**>>** user.valid?

=> false

Here **user.save** returns **false**, indicating a failed save. In the final command, we use the**valid?** method, which returns **false** when the object fails one or more validations, and **true**when all validations pass. (Recall from [Section 4.2.3](http://ruby.railstutorial.org/chapters/rails-flavored-ruby#sec:objects_and_message_passing) that Ruby uses a question mark to indicate such true/false *boolean* methods.) In this case, we only have one validation, so we know which one failed, but it can still be helpful to check using the **errors** object generated on failure:

**>>** user.errors.full\_messages

=> ["Name can't be blank"]