

---

## ValidatesTimeliness

- Source: [https://github.com/adzap/validates\\_timeliness](https://github.com/adzap/validates_timeliness)
- Issues: [https://github.com/adzap/validates\\_timeliness/issues](https://github.com/adzap/validates_timeliness/issues)

### Description

Complete validation of dates, times and datetimes for Rails 7.x and ActiveRecord.

Older Rails versions:

- Rails 4.x: [[https://github.com/adzap/validates\\_timeliness/tree/4-0-stable](https://github.com/adzap/validates_timeliness/tree/4-0-stable)]
- Rails 5.x: [[https://github.com/adzap/validates\\_timeliness/tree/5-0-stable](https://github.com/adzap/validates_timeliness/tree/5-0-stable)]
- Rails 6.x: [[https://github.com/adzap/validates\\_timeliness/tree/6-0-stable](https://github.com/adzap/validates_timeliness/tree/6-0-stable)]

### Features

- Adds validation for dates, times and datetimes to ActiveRecord
- Handles timezones and type casting of values for you
- Only Rails date/time validation plugin offering complete validation (See ORM/ODM support)
- Uses extensible date/time parser (Using `timeliness` gem. See `Plugin Parser`)
- Adds extensions to fix Rails date/time select issues (See `Extensions`)
- Supports I18n for the error messages. For multi-language support try `timeliness-i18n` gem.
- Supports all the Rubies (that any sane person would be using in production).

### Installation

In Gemfile

```
1 gem 'validates_timeliness', '~> 7.0.0.beta1'
```

Run bundler:

```
1 $ bundle install
```

Then run

```
1 $ rails generate validates_timeliness:install
```

This creates configuration initializer and locale files. In the initializer, there are a number of config options to customize the plugin.

---

**NOTE:** You may wish to enable the plugin parser and the extensions to start. Please read those sections first.

## Examples

```
1 validates_datetime :occurred_at
2
3 validates_date :date_of_birth, before: lambda { 18.years.ago },
4                               before_message: "must be at least 18
5                               years old"
6
7 validates_datetime :finish_time, after: :start_time # Method symbol
8
9 validates_date :booked_at, on: :create, on_or_after: :today # See
10 Restriction Shorthand.
11
12 validates_time :booked_at, between: ['9:00am', '5:00pm'] # On or after
13 9:00AM and on or before 5:00PM
14 validates_time :booked_at, between: '9:00am'..'5:00pm' # The same as
15 previous example
16 validates_time :booked_at, between: '9:00am'...'5:00pm' # On or after
17 9:00AM and strictly before 5:00PM
18
19 validates_time :breakfast_time, on_or_after: '6:00am',
20                               on_or_after_message: 'must be after
21                               opening time',
22                               before: :lunchtime,
23                               before_message: 'must be before lunch
24                               time'
```

## Usage

To validate a model with a date, time or datetime attribute you just use the validation method:

```
1 class Person < ActiveRecord::Base
2   validates_date :date_of_birth, on_or_before: lambda { Date.current }
3   # or
4   validates :date_of_birth, timeliness: { on_or_before: lambda { Date.
5     current }, type: :date }
6 end
```

or even on a specific record, per ActiveModel API.

```
1 @person.validates_date :date_of_birth, on_or_before: lambda { Date.
2   current }
```

---

The list of validation methods available are as follows: - `validates_date` - validate value as date - `validates_time` - validate value as time only i.e. '12:20pm' - `validates_datetime` - validate value as a full date and time - `validates` - use the `:timeliness` key and set the type in the hash.

The validation methods take the usual options plus some specific ones to restrict the valid range of dates or times allowed

Temporal options (or restrictions): - `:is_at` - Attribute must be equal to value to be valid - `:before` - Attribute must be before this value to be valid - `:on_or_before` - Attribute must be equal to or before this value to be valid - `:after` - Attribute must be after this value to be valid - `:on_or_after` - Attribute must be equal to or after this value to be valid - `:between` - Attribute must be between the values to be valid. Range or Array of 2 values.

Regular validation options: - `:allow_nil` - Allow a nil value to be valid - `:allow_blank` - Allows a nil or empty string value to be valid - `:if` - Execute validation when `:if` evaluates true - `:unless` - Execute validation when `:unless` evaluates false - `:on` - Specify validation context e.g `:save`, `:create` or `:update`. Default is `:save`.

Special options: - `:ignore_usec` - Ignores microsecond value on datetime restrictions - `:format` - Limit validation to a single format for special cases. Requires plugin parser.

The temporal restrictions can take 4 different value types: - Date, Time, or DateTime object value - Proc or lambda object which may take an optional parameter, being the record object - A symbol matching a method name in the model - String value

When an attribute value is compared to temporal restrictions, they are compared as the same type as the validation method type. So using `validates_date` means all values are compared as dates.

## Configuration

### ORM/ODM Support

The plugin adds date/time validation to ActiveRecord for any ORM/ODM that supports the ActiveRecord validations component. However, there is an issue with most ORM/ODMs which does not allow 100% date/time validation by default. Specifically, when you assign an invalid date/time value to an attribute, most ORM/ODMs will only store a nil value for the attribute. This causes an issue for date/time validation, since we need to know that a value was assigned but was invalid. To fix this, we need to cache the original invalid value to know that the attribute is not just nil.

Each ORM/ODM requires a specific shim to fix it. The plugin includes a shim for ActiveRecord and Mongoid. You can activate them like so

```
1 ValidatesTimeliness.setup do |config|
```

---

```
2 # Extend ORM/ODMs for full support (:active_record).
3 config.extend_orms = [ :active_record ]
4 end
```

By default the plugin extends ActiveRecord if loaded. If you wish to extend another ORM then look at the wiki page for more information.

It is not required that you use a shim, but you will not catch errors when the attribute value is invalid and evaluated to nil.

## Error Messages

Using the I18n system to define new defaults:

```
1 en:
2   errors:
3     messages:
4       invalid_date: "is not a valid date"
5       invalid_time: "is not a valid time"
6       invalid_datetime: "is not a valid datetime"
7       is_at: "must be at %{restriction}"
8       before: "must be before %{restriction}"
9       on_or_before: "must be on or before %{restriction}"
10      after: "must be after %{restriction}"
11      on_or_after: "must be on or after %{restriction}"
```

The `%{restriction}` signifies where the interpolation value for the restriction will be inserted.

You can also use validation options for custom error messages. The following option keys are available:

```
1 :invalid_date_message
2 :invalid_time_message
3 :invalid_datetime_message
4 :is_at_message
5 :before_message
6 :on_or_before_message
7 :after_message
8 :on_or_after_message
```

**Note:** There is no `:between_message` option. The between error message should be defined using the `:on_or_after` and `:on_or_before` (`:before` in case when `:between` argument is a `Range` with excluded high value, see Examples) messages.

It is highly recommended you use the I18n system for error messages.

---

## Plugin Parser

The plugin uses the `timeliness` gem as a fast, configurable and extensible date and time parser. You can add or remove valid formats for `dates`, `times`, and `datetimes`. It is also more strict than the Ruby parser, which means it won't accept day of the month if it's not a valid number for the month.

By default the parser is disabled. To enable it:

```
1 # in the setup block
2 config.use_plugin_parser = true
```

Enabling the parser will mean that strings assigned to attributes validated with the plugin will be parsed using the gem. See the wiki for more details about the parser configuration.

## Restriction Shorthand

It is common to restrict an attribute to being on or before the current time or current day. To specify this you need to use a lambda as an option value e.g. `lambda { Time.current }`. This can be tedious noise amongst your validations for something so common. To combat this the plugin allows you to use shorthand symbols for often used relative times or dates.

Just provide the symbol as the option value like so:

```
1 validates_date :birth_date, on_or_before: :today
```

The `:today` symbol is evaluated as `lambda { Date.current }`. The `:now` and `:today` symbols are pre-configured. Configure your own like so:

```
1 # in the setup block
2 config.restriction_shorthand_symbols.update(yesterday: lambda { 1.day.ago })
```

## Default Timezone

The plugin needs to know the default timezone you are using when parsing or type casting values. If you are using ActiveRecord then the default is automatically set to the same default zone as ActiveRecord. If you are using another ORM you may need to change this setting.

```
1 # in the setup block
2 config.default_timezone = :utc
```

By default it will be `UTC` if ActiveRecord is not loaded.

---

## Dummy Date For Time Types

Given that Ruby has no support for a time-only type, all time type columns are evaluated as a regular Time class objects with a dummy date value set. Rails defines the dummy date as 2000-01-01. So a time of '12:30' is evaluated as a Time value of '2000-01-01 12:30'. If you need to customize this for some reason you can do so as follows:

```
1 # in the setup block
2 config.dummy_date_for_time_type = [2009, 1, 1]
```

The value should be an array of 3 values being year, month and day in that order.

## Temporal Restriction Errors

When using the validation temporal restrictions there are times when the restriction option value itself may be invalid. This will add an error to the model such as 'Error occurred validating birth\_date for :before restriction'. These can be annoying in development or production as you most likely just want to skip the option if no valid value was returned. By default these errors are displayed in Rails test mode.

To turn them on/off:

```
1 # in the setup block
2 config.ignore_restriction_errors = true
```

## Extensions

### Strict Parsing for Select Helpers

When using date/time select helpers, the component values are handled by ActiveRecord using the Time class to instantiate them into a time value. This means that some invalid dates, such as 31st June, are shifted forward and treated as valid. To handle these cases in a strict way, you can enable the plugin extension to treat them as invalid dates.

To activate it, uncomment this line in the initializer:

```
1 # in the setup block
2 config.enable_multiparameter_extension!
```

---

## Display Invalid Values in Select Helpers

The plugin offers an extension for ActionView to allowing invalid date and time values to be redisplayed to the user as feedback, instead of a blank field which happens by default in Rails. Though the date helpers make this a pretty rare occurrence, given the select dropdowns for each date/time component, but it may be something of interest.

To activate it, uncomment this line in the initializer:

```
1 # in the setup block
2 config.enable_date_time_select_extension!
```

## Contributors

To see the generous people who have contributed code, take a look at the contributors list.

## Maintainers

- Adam Meehan

## License

Copyright (c) 2021 Adam Meehan, released under the MIT license.