## NAME

Catalyst::Manual::Tutorial::09_AdvancedCRUD::09_FormFu - Catalyst Tutorial - Chapter 9: Advanced CRUD - FormFu

## OVERVIEW

This is **Chapter 9 of 10** for the Catalyst tutorial.

Tutorial Overview

1. Introduction
2. Catalyst Basics
3. More Catalyst Basics
4. Basic CRUD
5. Authentication
6. Authorization
7. Debugging
8. Testing
9. **09_Advanced CRUD::09_FormFu**
10. Appendices

## DESCRIPTION

This portion of the tutorial explores HTML::FormFu and how it can be used to manage forms, perform validation of form input, as well as save and restore data to/from the database.

See Catalyst::Manual::Tutorial::09_AdvancedCRUD for additional form management options other than HTML::FormFu.

Source code for the tutorial in included in the */home/catalyst/Final* directory of the Tutorial Virtual machine (one subdirectory per chapter). There are also instructions for downloading the code in Catalyst::Manual::Tutorial::01_Intro.

## HTML::FormFu FORM CREATION

This section looks at how HTML::FormFu can be used to add additional functionality to the manually created form from Chapter 4.

### Inherit From Catalyst::Controller::HTML::FormFu

First, change your `lib/MyApp/Controller/Books.pm` to inherit from Catalyst::Controller::HTML::FormFu by changing the `extends` line from the default of:

```
    BEGIN {extends 'Catalyst::Controller'; }
```

to use the FormFu base controller class:

```
    BEGIN {extends 'Catalyst::Controller::HTML::FormFu'; }
```

Don't forget to add:

```
    requires 'HTML::FormFu';
    requires 'Catalyst::Controller::HTML::FormFu';
    requires 'requires 'HTML::FormFu::Model::DBIC';';
```

to your `Makefile.PL`.

### Add Action to Display and Save the Form

Open `lib/MyApp/Controller/Books.pm` in your editor and add the following method:

```
    =head2 formfu_create

    Use HTML::FormFu to create a new book

    =cut

    sub formfu_create :Chained('base') :PathPart('formfu_create') :Args(0) :FormConfig {
        my ($self, $c) = @_;

        # Get the form that the :FormConfig attribute saved in the stash
        my $form = $c->stash->{form};

        # Check if the form has been submitted (vs. displaying the initial
        # form) and if the data passed validation.  "submitted_and_valid"
        # is shorthand for "$form->submitted && !$form->has_errors"
        if ($form->submitted_and_valid) {
            # Create a new book
            my $book = $c->model('DB::Book')->new_result({});
            # Save the form data for the book
            $form->model->update($book);
            # Set a status message for the user & return to books list
            $c->response->redirect($c->uri_for($self->action_for('list'),
```

```
            {mid => $c->set_status_msg("Book created")}));
        $c->detach;
    } else {
        # Get the authors from the DB
        my @author_objs = $c->model("DB::Author")->all();
        # Create an array of arrayrefs where each arrayref is an author
        my @authors;
        foreach (sort {$a->last_name cmp $b->last_name} @author_objs) {
            push(@authors, [$_->id, $_->last_name]);
        }
        # Get the select added by the config file
        my $select = $form->get_element({type => 'Select'});
        # Add the authors to it
        $select->options(\@authors);
    }

    # Set the template
    $c->stash(template => 'books/formfu_create.tt2');
}
```

## Create a Form Config File

Although `HTML::FormFu` supports any configuration file handled by Config::Any, most people tend to use YAML. First create a directory to hold your form configuration files:

```
$ mkdir -p root/forms/books
```

Then create the file `root/forms/books/formfu_create.yml` and enter the following text:

```
---
# indicator is the field that is used to test for form submission
indicator: submit
# Start listing the form elements
elements:
    # The first element will be a text field for the title
    - type: Text
      name: title
      label: Title
      # This is an optional 'mouse over' title pop-up
      attributes:
        title: Enter a book title here

    # Another text field for the numeric rating
    - type: Text
      name: rating
      label: Rating
      attributes:
        title: Enter a rating between 1 and 5 here

    # Add a drop-down list for the author selection.  Note that we will
    # dynamically fill in all the authors from the controller but we
    # could manually set items in the drop-list by adding this YAML code:
    # options:
    #   - [ '1', 'Bastien' ]
    #   - [ '2', 'Nasseh'  ]
    - type: Select
      name: authors
      label: Author

    # The submit button
```

```
        - type: Submit
          name: submit
          value: Submit
```

**NOTE:** Copying and pasting YAML from Perl documentation is sometimes tricky. See the "Config::General Config for this tutorial" section of this document for a more foolproof config format.

### Update the CSS

Edit `root/static/css/main.css` and add the following lines to the bottom of the file:

```
...
input {
    display: block;
}
select {
    display: block;
}
.submit {
    padding-top: .5em;
    display: block;
}
```

These changes will display form elements vertically.

### Create a Template Page To Display The Form

Open `root/src/books/formfu_create.tt2` in your editor and enter the following:

```
[% META title = 'Create/Update Book' %]

[%# Render the HTML::FormFu Form %]
[% form %]

<p><a href="[% c.uri_for(c.controller.action_for('list'))
    %]">Return to book list</a></p>
```

### Add Links for Create and Update via `HTML::FormFu`

Open `root/src/books/list.tt2` in your editor and add the following to the bottom of the existing file:

```
...
<p>
  HTML::FormFu:
  <a href="[% c.uri_for(c.controller.action_for('formfu_create')) %]">Create</a>
</p>
```

This adds a new link to the bottom of the book list page that we can use to easily launch our HTML::FormFu-based form.

### Test The HTML::FormFu Create Form

Make sure the server is running with the "-r" restart option:

```
$ script/myapp_server.pl -r
```

Login as `test01` (password: mypass). Once at the Book List page, click the new HTML::FormFu "Create" link at the bottom to display the form. Fill in the following values:

```
Title:  Internetworking with TCP/IP Vol. II
Rating: 4
Author: Comer
```

Click the "Submit" button, and you will be returned to the Book List page with a "Book created" status message displayed.

Also note that this implementation allows you to create books with any bogus information. Although we have constrained the authors with the drop-down list (note that this isn't bulletproof because we still have not prevented a user from "hacking" the form to specify other values), there are no restrictions on items such as the length of the title (for example, you can create a one-letter title) and the value of the rating (you can use any number you want, and even non-numeric values with SQLite). The next section will address this concern.

**Note:** Depending on the database you are using and how you established the columns in your tables, the database could obviously provide various levels of "type enforcement" on your data. The key point being made in the previous paragraph is that the *web application* itself is not performing any validation.

## HTML::FormFu VALIDATION AND FILTERING

Although the use of HTML::FormFu in the previous section did provide an automated mechanism to build the form, the real power of this module stems from functionality that can automatically validate and filter the user input. Validation uses constraints to be sure that users input appropriate data (for example, that the email field of a form contains a valid email address). Filtering can also be used to remove extraneous whitespace from fields or to escape meta-characters in user input.

### Add Constraints

Open `root/forms/books/formfu_create.yml` in your editor and update it to match:

```
---
# indicator is the field that is used to test for form submission
indicator: submit
# Start listing the form elements
elements:
    # The first element will be a text field for the title
    - type: Text
      name: title
      label: Title
      # This is an optional 'mouse over' title pop-up
      attributes:
        title: Enter a book title here
      # Add constraints for the field
      constraints:
        # Force the length to be between 5 and 40 chars
        - type: Length
          min: 5
          max: 40
          # Override the default of 'Invalid input'
```

```
            message: Length must be between 5 and 40 characters

    # Another text field for the numeric rating
    - type: Text
      name: rating
      label: Rating
      attributes:
        title: Enter a rating between 1 and 5 here
      # Use Filter to clean up the input data
      # Could use 'NonNumeric' below, but since Filters apply *before*
      # constraints, it would conflict with the 'Integer' constraint below.
      # So let's skip this and just use the constraint.
      #filter:
        # Remove everything except digits
        #- NonNumeric
      # Add constraints to the field
      constraints:
        # Make sure it's a number
        - type: Integer
          message: "Required. Digits only, please."
        # Check the min & max values
        - type: Range
          min: 1
          max: 5
          message: "Must be between 1 and 5."

    # Add a select list for the author selection.  Note that we will
    # dynamically fill in all the authors from the controller but we
    # could manually set items in the select by adding this YAML code:
    # options:
    #   - [ '1', 'Bastien' ]
    #   - [ '2', 'Nasseh'  ]
    - type: Select
      name: authors
      label: Author
      # Convert the drop-down to a multi-select list
      multiple: 1
      # Display 3 entries (user can scroll to see others)
      size: 3
      # One could argue we don't need to do filters or constraints for
      # a select list, but it's smart to do validation and sanity
      # checks on this data in case a user "hacks" the input
      # Add constraints to the field
      constraints:
        # Make sure it's a number
        - Integer

    # The submit button
    - type: Submit
      name: submit
      value: Submit

# Global filters and constraints.
constraints:
    # The user cannot leave any fields blank
    - Required
    # If not all fields are required, move the Required constraint to the
    # fields that are
filter:
    # Remove whitespace at both ends
    - TrimEdges
    # Escape HTML characters for safety
    - HTMLEscape
```

**NOTE:** Copying and pasting YAML from Perl documentation is sometimes tricky. See the "Config::General Config for this tutorial" section of this document for a more foolproof config format.

The main changes are:

- The `Select` element for `authors` is changed from a single-select drop-down to a multi-select list by adding configuration for the `multiple` and `size` options in `formfu_create.yml`.
- Constraints are added to provide validation of the user input. See HTML::FormFu::Constraint for other constraints that are available.
- A variety of filters are run on every field to remove and escape unwanted input. See HTML::FormFu::Filter for more filter options.

### Try Out the Updated Form

Make sure you are still logged in as `test01` and try adding a book with various errors: title less than 5 characters, non-numeric rating, a rating of 0 or 6, etc. Also try selecting one, two, and zero authors. When you click Submit, the HTML::FormFu `constraint` items will validate the logic and insert feedback as appropriate. Try adding blank spaces at the front or the back of the title and note that it will be removed.

Note that you can update your FormFu YAML forms and the development server does not need to reload -- the form definition is read from the YAML file each time a controller action uses it.

## CREATE AND UPDATE/EDIT ACTION

Let's expand the work done above to add an edit action. First, open `lib/MyApp/Controller/Books.pm` and add the following method to the bottom:

```
    =head2 formfu_edit

    Use HTML::FormFu to update an existing book

    =cut

    sub formfu_edit :Chained('object') :PathPart('formfu_edit') :Args(0)
            :FormConfig('books/formfu_create.yml') {
        my ($self, $c) = @_;

        # Get the specified book already saved by the 'object' method
        my $book = $c->stash->{object};

        # Make sure we were able to get a book
        unless ($book) {
            # Set an error message for the user & return to books list
            $c->response->redirect($c->uri_for($self->action_for('list'),
                {mid => $c->set_error_msg("Invalid book -- Cannot edit")}));
            $c->detach;
        }

        # Get the form that the :FormConfig attribute saved in the stash
        my $form = $c->stash->{form};

        # Check if the form has been submitted (vs. displaying the initial
        # form) and if the data passed validation.  "submitted_and_valid"
        # is shorthand for "$form->submitted && !$form->has_errors"
        if ($form->submitted_and_valid) {
            # Save the form data for the book
            $form->model->update($book);
            # Set a status message for the user
```

```
            # Set a status message for the user & return to books list
            $c->response->redirect($c->uri_for($self->action_for('list'),
                {mid => $c->set_status_msg("Book edited")}));
            $c->detach;
        } else {
            # Get the authors from the DB
            my @author_objs = $c->model("DB::Author")->all();
            # Create an array of arrayrefs where each arrayref is an author
            my @authors;
            foreach (sort {$a->last_name cmp $b->last_name} @author_objs) {
                push(@authors, [$_->id, $_->last_name]);
            }
            # Get the select added by the config file
            my $select = $form->get_element({type => 'Select'});
            # Add the authors to it
            $select->options(\@authors);
            # Populate the form with existing values from DB
            $form->model->default_values($book);
        }

        # Set the template
        $c->stash(template => 'books/formfu_create.tt2');
    }
```

Most of this code should look familiar to what we used in the `formfu_create` method (in fact, we should probably centralize some of the common code in separate methods). The main differences are:

- We have to manually specify the name of the FormFu .yml file as an argument to `:FormConfig` because the name can no longer be automatically deduced from the name of our action/method (by default, FormFu would look for a file named `books/formfu_edit.yml`).
- We load the book object from the stash (found using the $id passed to the Chained object method)
- We use `$id` to look up the existing book from the database.
- We make sure the book lookup returned a valid book. If not, we set the error message and return to the book list.
- If the form has been submitted and passes validation, we skip creating a new book and just use `$form->model->update` to update the existing book.
- If the form is being displayed for the first time (or has failed validation and it being redisplayed), we use `$form->model->default_values` to populate the form with data from the database.

Then, edit `root/src/books/list.tt2` and add a new link below the existing "Delete" link that allows us to edit/update each existing book. The last <td> cell in the book list table should look like the following:

```
...
<td>
  [% # Add a link to delete a book %]
  <a href="[%
    c.uri_for(c.controller.action_for('delete'), [book.id]) %]">Delete</a>
  [% # Add a link to edit a book %]
  <a href="[%
    c.uri_for(c.controller.action_for('formfu_edit'), [book.id]) %]">Edit</a>
</td>
...
```

**Note:** Only add three lines (the "Add a link to edit a book" comment and the href for `formfu_edit`). Make sure you add it below the existing `delete` link.

### Try Out the Edit/Update Feature

Make sure you are still logged in as `test01` and go to the http://localhost:3000/books/list URL in your browser. Click the "Edit" link next to "Internetworking with TCP/IP Vol. II", change the rating to a 3, the "II" at end of the title to the number "2", add Stevens as a co-author (control-click), and click Submit. You will then be returned to the book list with a "Book edited" message at the top in green. Experiment with other edits to various books.

### More Things to Try

You are now armed with enough knowledge to be dangerous. You can keep tweaking the example application; some things you might want to do:

- Add an appropriate authorization check to the new Edit function.
- Cleanup the List page so that the Login link only displays when the user isn't logged in and the Logout link only displays when a user is logged in.
- Add a more sensible policy for when and how users and admins can do things in the CRUD cycle.
- Support the CRUD cycle for authors.

Or you can proceed to write your own application, which is probably the real reason you worked through this Tutorial in the first place.

### Config::General Config for this tutorial

If you are having difficulty with YAML config above, please save the below into the file `formfu_create.conf` and delete the `formfu_create.yml` file. The below is in Config::General format which follows the syntax of Apache config files.

```
    constraints   Required
    <elements>
        <constraints>
            min   5
            max   40
            type   Length
            message   Length must be between 5 and 40 characters
        </constraints>
        filter   TrimEdges
        filter   HTMLEscape
        name   title
        type   Text
        label   Title
        <attributes>
            title   Enter a book title here
        </attributes>
    </elements>
    <elements>
        constraints   Integer
        filter   TrimEdges
        filter   NonNumeric
        name   rating
        type   Text
        label   Rating
        <attributes>
            title   Enter a rating between 1 and 5 here
        </attributes>
    </elements>
    <elements>
        constraints   Integer
        filter   TrimEdges
        filter   HTMLEscape
```

```
        name   authors
        type   Select
        label   Author
        multiple   1
        size   3
    </elements>
    <elements>
        value   Submit
        name   submit
        type   Submit
    </elements>
    indicator   submit
```

## AUTHOR

Kennedy Clark, hkclark@gmail.com

Feel free to contact the author for any errors or suggestions, but the best way to report issues is via the CPAN RT Bug system at https://rt.cpan.org/Public/Dist/Display.html?Name=Catalyst-Manual.