tutorial.todoapp Documentation

Release

June 23, 2014

Contents

1 The Tutorial **3**

1.1 Prerequisites 3
1.2 Tutorial Setup 3
1.3 Virtualenv 7
1.4 Troubleshooting 7
1.5 Chapter 1: Through-The-Web 8
1.6 Chapter 2: Filesystem package 29
1.7 Chapter 3: Custom View 37
1.8 Chapter 4: Bling-bling 40

2 Developer Documentation **45**

2.1 Releasing a new version 45
2.2 Changelog 46
2.3 License (3-clause BSD) 47

3 Indices and tables **49**

WARNING: If you are reading this on GitHub, DON'T! Read it on ReadTheDocs:
http://tutorialtodoapp.readthedocs.org/en/latest/index.html so you have
working references and proper formatting.

Framework Plone 4.3

Bug tracker https://github.com/collective/tutorial.todoapp/issues

Source https://github.com/collective/tutorial.todoapp

Documentation http://tutorialtodoapp.readthedocs.org/

Code status

You will learn to:

- create custom content-types Through-The-Web
- create and apply custom workflows
- create custom listings
- dump your changes into a filesytem package to future-proof them
- write tests for your filesystem package

Summary

It's a fact - Plone has a lot of complicated features. That doesn't mean Plone is hard for everything! This is a simple tutorial that anyone can follow to get a simple Todo list application running inside of Plone. Would you want to deploy Plone for just a Todo list in real life? Probably not. You can however learn several simple, fast concepts that will get you most of the way there. Feeling like you don't understand something completely or the terminology is getting to you? Sit back, relax, and finish the tutorial. If in the end things still aren't clear, please give feedback and we'll look at what we could do better.

CHAPTER 1

The Tutorial

WARNING: If you are reading this on GitHub, DON'T! Read it on ReadTheDocs:
http://tutorialtodoapp.readthedocs.org/en/latest/prelude.html so you have
working references and proper formatting.

1.1 Prerequisites

- You have Git installed and vaguely know how to use it.
- You are working with Python 2.6 or 2.7
- You have already installed (listed names are for Ubuntu/Debian, should be similar for your distribution): *python-setuptools*, *python-virtualenv*, *zlib1g-dev*, *libxslt1-dev* and *libxml2-dev*.
- For Ubuntu/Debian users it may be worthwhile to install build-essential (sudo apt-get install build-essential) to make sure you have necessary build tools.
- Sorry Windows users, but you'll have to translate as usual from n*x to Windows-ese.

1.2 Tutorial Setup

Since this is a tutorial on how to be a developer, there will always be a little bit of setup. There are many ways that this could be done and integrated with the Plone Unified Installer, but those are not covered here. It is possible though to use this tutorial in the context of the Unified Installer by just installing the source skeleton.

1. Using Git, checkout the project code for this tutorial. Then run `make` to prepare the development environment. There are sometimes problems on Mac and Linux machines with pre-installed versions of Python. If you run into issues, please see *Troubleshooting*:

```
> mkdir tutorial.todoapp
> git clone git://github.com/collective/tutorial.todoapp.git ./
> make
```

Note: This will take your system python interpreter. If you wish to use a custom one, run it like *make python=/usr/local/bin/python2.7*

Note: Whenever you are stuck with a broken environment and want to start over, run `make clean` followed by `make`. This will remove everything but your source files and your database, and then continue to rebuild the entire environment.

Note: Running `make` also generates this documentation for you locally and runs all tests. See `Makefile` for other commands you have available, such as `make docs` and `make tests`.

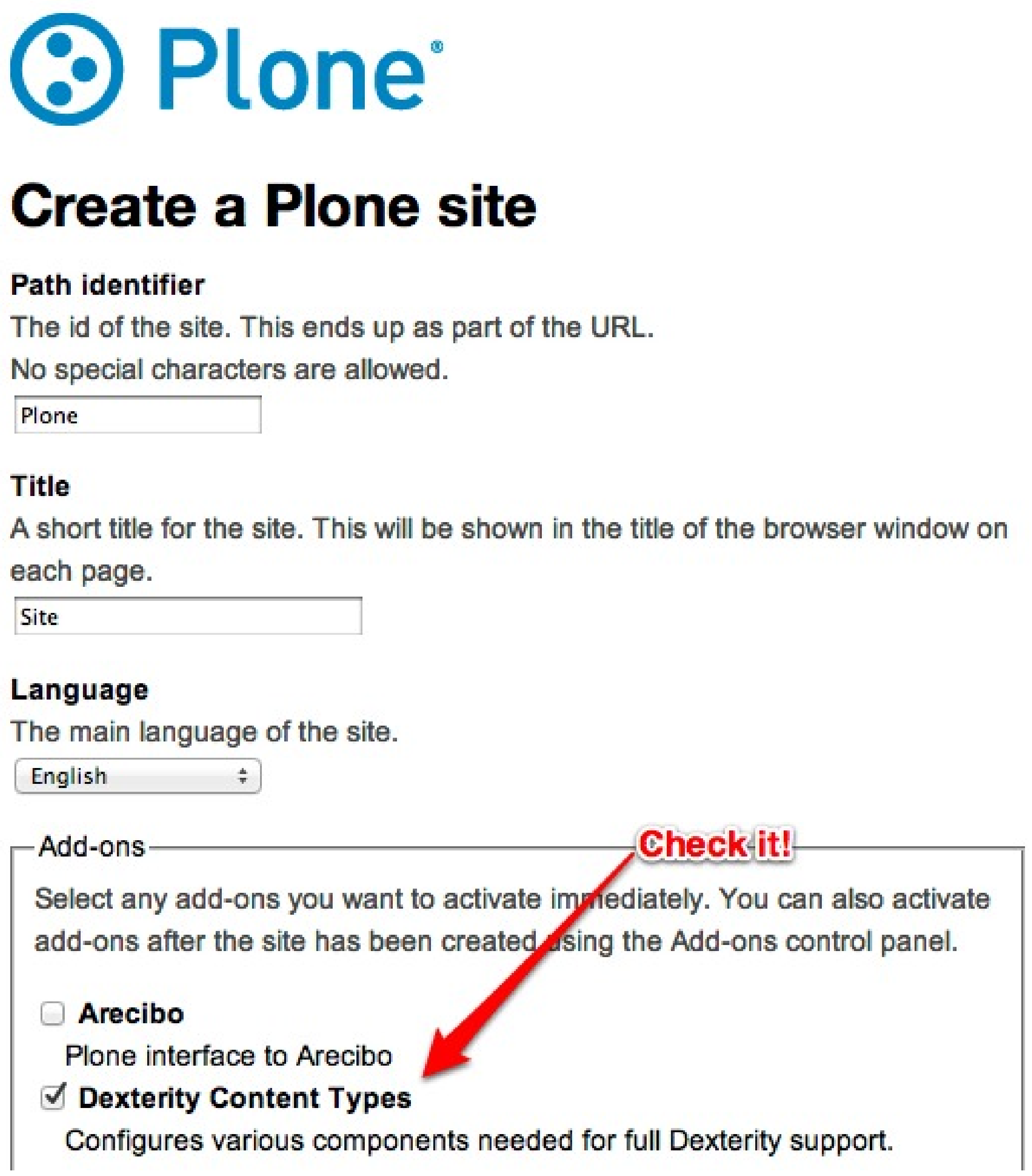
1. Before starting the Plone instance, lets activate our virtualenv. For more information on virtualenv check *Virtualenv*:

   ```
   > source bin/activate
   ```

2. Next up, start the Plone instance:

   ```
   > ./bin/instance fg
   ```

3. Open up your browser and navigate to `http://localhost:8080/`
4. Click ‘Create a New Plone Site’. The default username and password is `admin:admin`.
5. Change the *name* and *id* if you wish, but keep in mind that for this tutorial we will assume that the name of the site is `Plone` and the Plone instance is located at `http://localhost:8080/Plone`.



Plone

Create a Plone site

Path identifier

The id of the site. This ends up as part of the URL.
No special characters are allowed.

Plone

Title

A short title for the site. This will be shown in the title of the browser window on each page.

Site

Language

The main language of the site.

English

Add-ons

Select any add-ons you want to activate immediately. You can also activate add-ons after the site has been created using the Add-ons control panel.

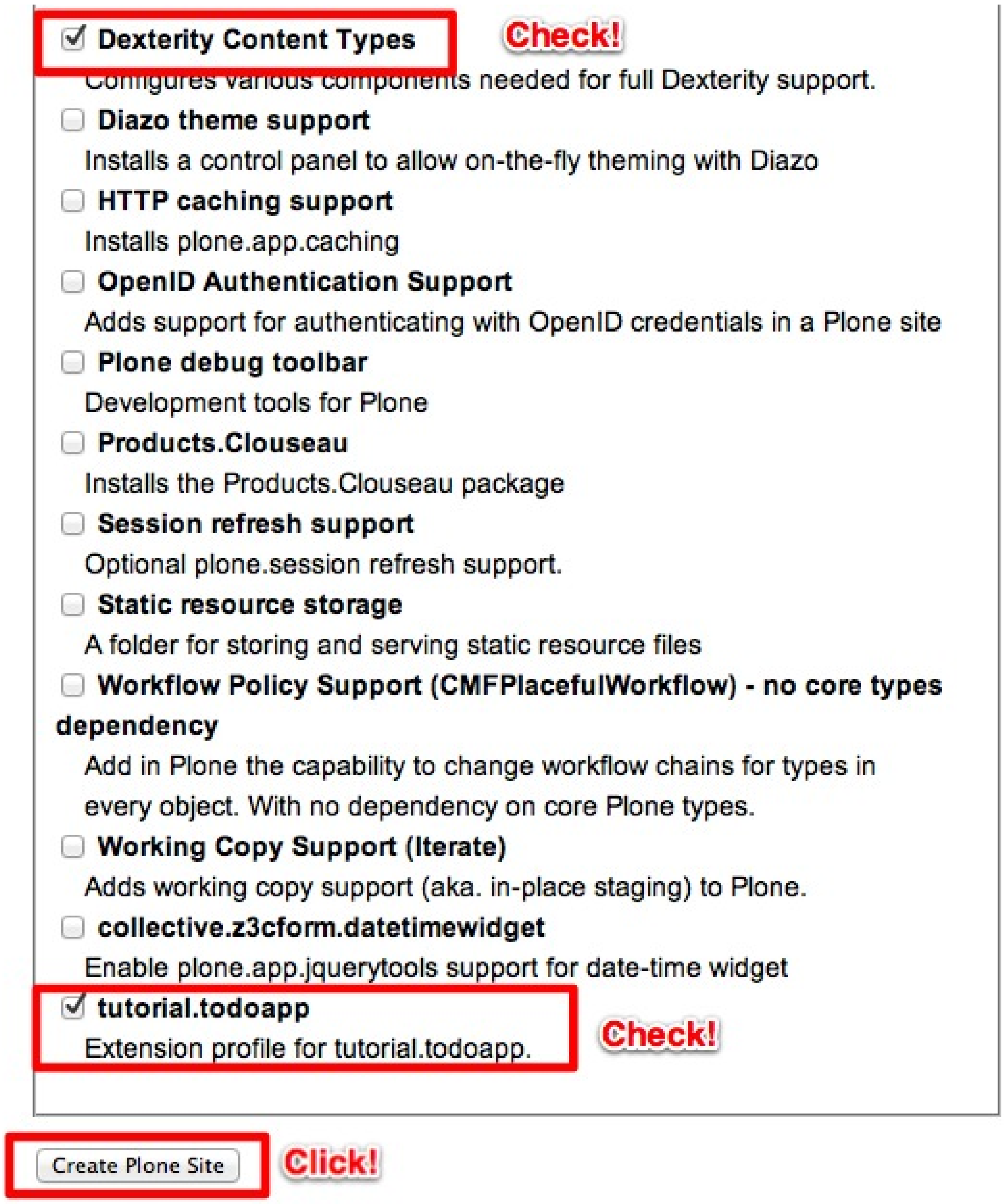
☐ **Arecibo**

Plone interface to Arecibo

✓ **Dexterity Content Types**

Configures various components needed for full Dexterity support.

6. Under *Add-ons*, make sure to check `Dexterity Content Types` and `tutorial.todoapp` then click `Create Plone Site`.



✓ **Dexterity Content Types**
Configures various components needed for full Dexterity support.

☐ **Diazo theme support**
Installs a control panel to allow on-the-fly theming with Diazo

☐ **HTTP caching support**
Installs plone.app.caching

☐ **OpenID Authentication Support**
Adds support for authenticating with OpenID credentials in a Plone site

☐ **Plone debug toolbar**
Development tools for Plone

☐ **Products.Clouseau**
Installs the Products.Clouseau package

☐ **Session refresh support**
Optional plone.session refresh support.

☐ **Static resource storage**
A folder for storing and serving static resource files

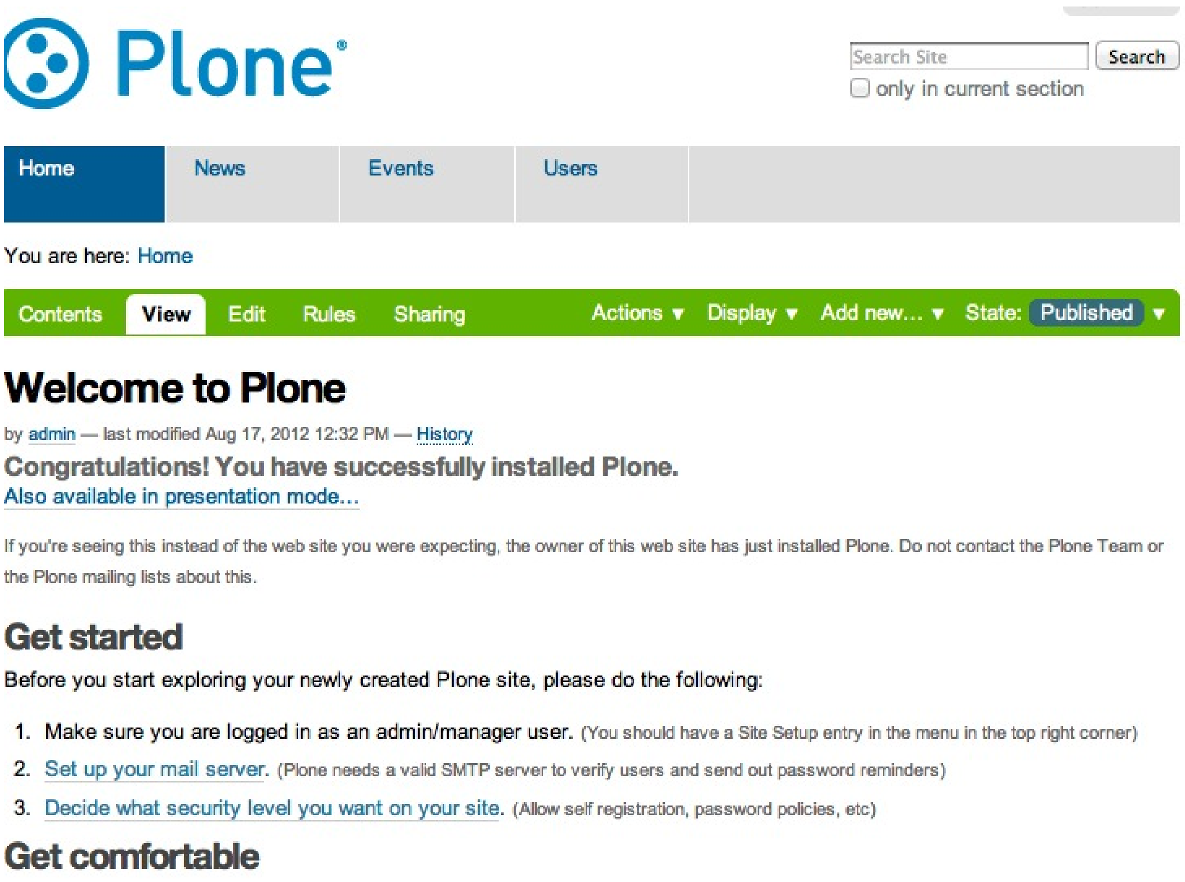
☐ **Workflow Policy Support (CMFPlacefulWorkflow) - no core types dependency**
Add in Plone the capability to change workflow chains for types in every object. With no dependency on core Plone types.

☐ **Working Copy Support (Iterate)**
Adds working copy support (aka. in-place staging) to Plone.

☐ **collective.z3cform.datetimewidget**
Enable plone.app.jquerytools support for date-time widget

✓ **tutorial.todoapp**
Extension profile for tutorial.todoapp.

7. There, your Plone site is created and you can continue with the tutorial.



Welcome to Plone

by admin — last modified Aug 17, 2012 12:32 PM — History

Congratulations! You have successfully installed Plone.

Also available in presentation mode…

If you're seeing this instead of the web site you were expecting, the owner of this web site has just installed Plone. Do not contact the Plone Team or the Plone mailing lists about this.

Get started

Before you start exploring your newly created Plone site, please do the following:

1. Make sure you are logged in as an admin/manager user. (You should have a Site Setup entry in the menu in the top right corner)
2. Set up your mail server. (Plone needs a valid SMTP server to verify users and send out password reminders)
3. Decide what security level you want on your site. (Allow self registration, password policies, etc)

Get comfortable

Woot! Let's go.

1.3 Virtualenv

virtualenv is a tool to create isolated Python environments. virtualenv documentation.

1.4 Troubleshooting

Sometimes setting up development environment gives you lemons. There are various ways to go around that.

In case you don't have correct Python version or your system Python environment is broken (yes, I'm looking to you OS X), *buildout.python* gives you get out of jail free card. To install it, see the install docs. Then use *buildout.python/python-2.7/bin/python bootstrap.py –distribute* step as in *Tutorial Setup* section and so on.

If everything fails, it's time to use a virtual machine. See install steps to prepare and try again with *Tutorial Setup* section.

WARNING: If you are reading this on GitHub, DON'T! Read it on ReadTheDocs:
http://tutorialtodoapp.readthedocs.org/en/latest/chapter_1.html so you
have working references and proper formatting.

1.5 Chapter 1: Through-The-Web

1.5.1 Getting Started with Content Types

If you don't know what a content type is, don't worry! Sit back, relax, and do the tutorial! I'll save the mumbo jumbo definitions for another day. In this first part, we will make a Todo list without touching any code. It won't be fancy, but it will give you a good idea of how things work in Plone.

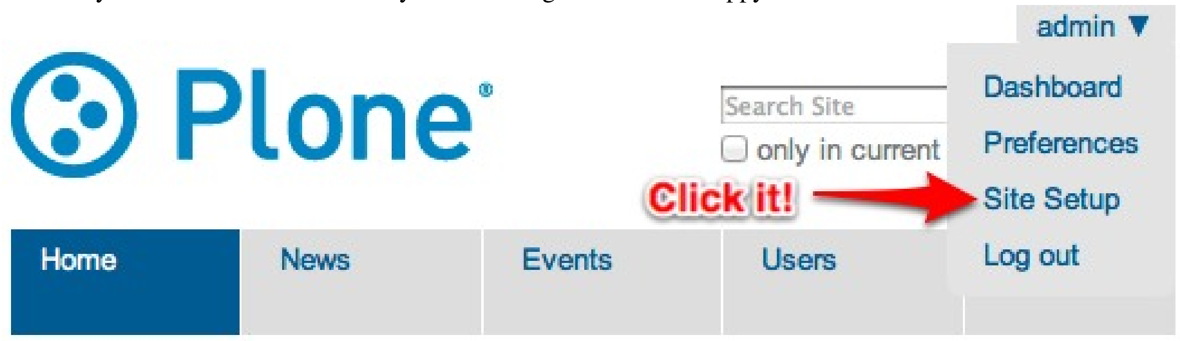
The way Plone handles content is a little different than your average relational database driven framework, so if you don't understand something right away, sit back, relax, and finish the tutorial.

Generally speaking, content-types are just that: types of content. By default, in Plone you get the News Item content-type, the Event content-type and so on. So if you add a content item that is of Event type, you are using the Event content-type. In our case, we will create a new content-type that will represent a Todo Item.

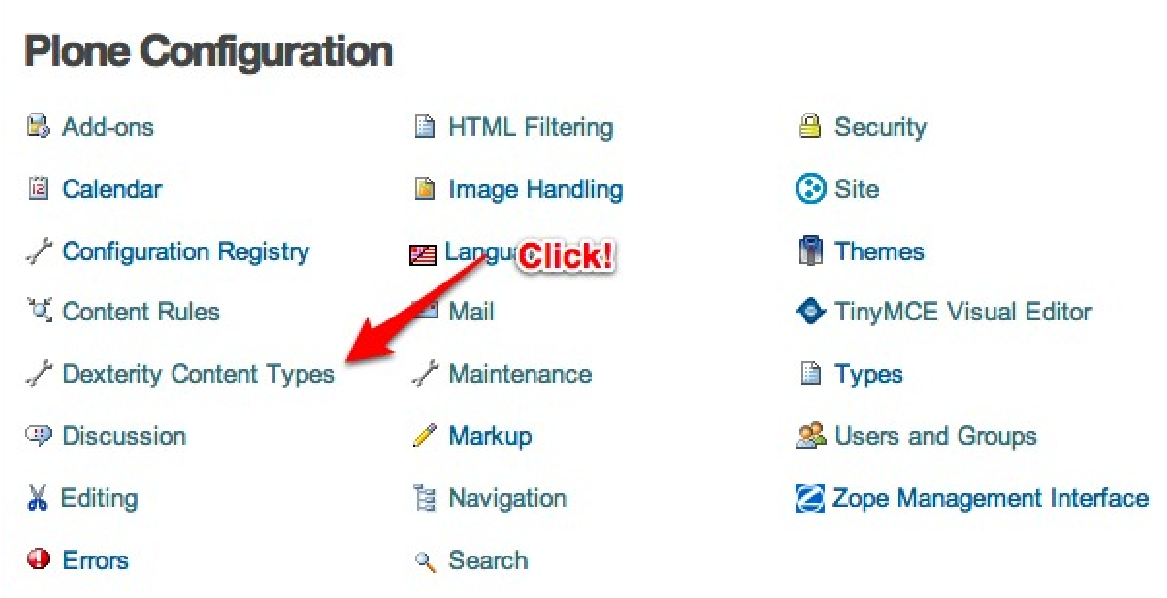
Create a New Content Type

First we need to create a new content type to represent an item on our Todo list. This will be a type with one field, that which needs to be done.

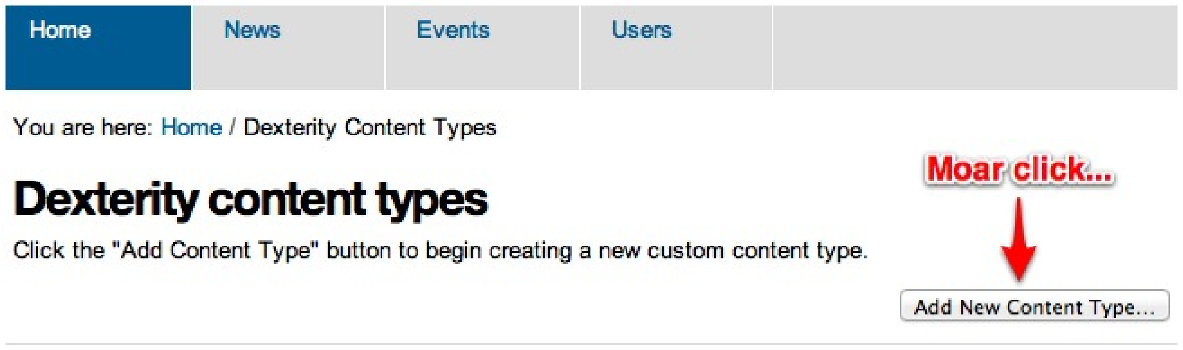
1. Navigate to *site setup* as shown below, or just enter `http://localhost:8080/Plone/@@overview-controlpanel` in your browser. This is where you can configure Plone for happy fun time.



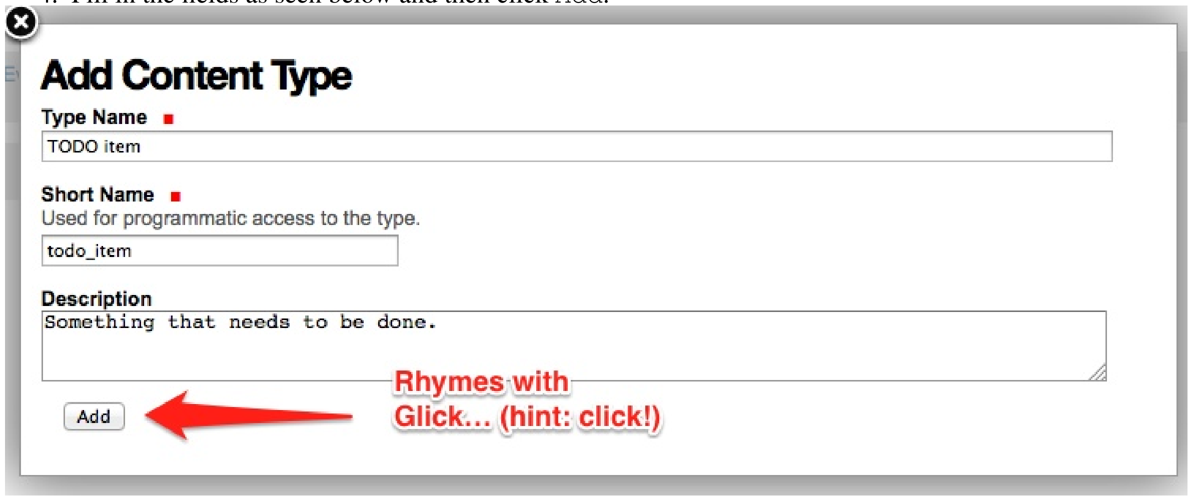
2. Now comes the fun part. We want to create our own type *Through-The-Web* aka. *TTW*. This type will be a Todo Item. Let's click *Dexterity Content Types* (or go directly to `http://localhost:8080/Plone/@@dexterity-types`).



3. Create a Todo List Item by clicking `Add New Content Type`.


Home
News
Events
Users
You are here: Home / Dexterity Content Types
Dexterity content types
Click the "Add Content Type" button to begin creating a new custom content type.
Moar click...
Add New Content Type...

4. Fill in the fields as seen below and then click `Add`.

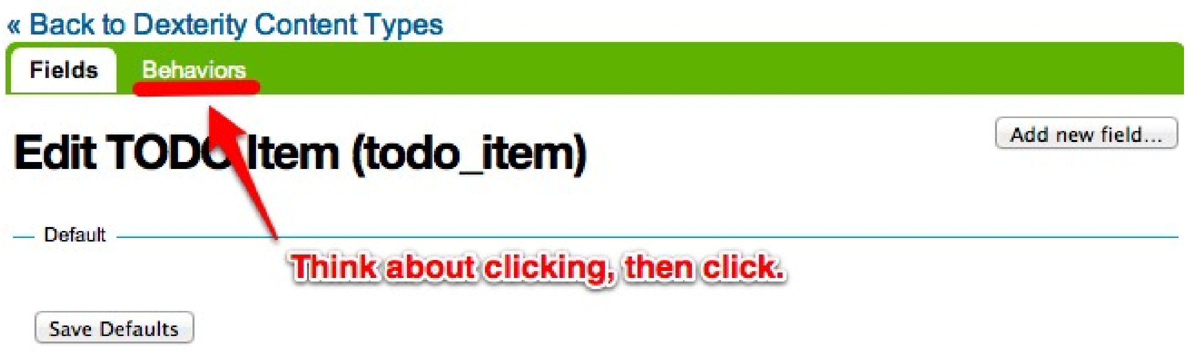


Type Name: TODO item

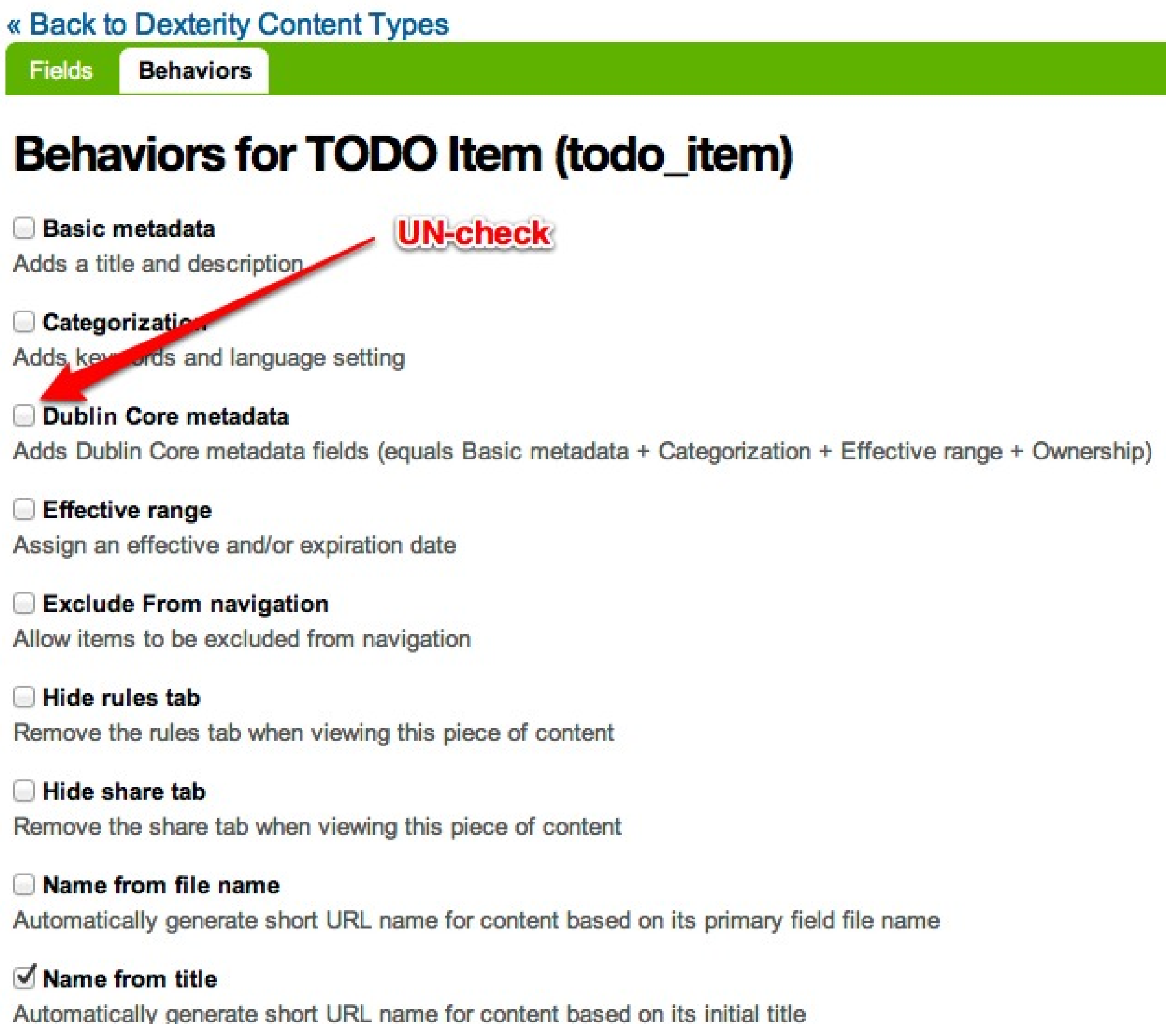
Short Name: Used for programmatic access to the type. todo_item

Description: Something that needs to be done.

5. Now you will see that there is a new type to play with. There are two important things we need to do here: we need to adjust some *behaviors*, and add some *fields*. Let's look at behaviors first.


« Back to Dexterity Content Types
Fields
Behaviors
Add new field...
Edit TODO Item (todo_item)
Default
Think about clicking, then click.
Save Defaults

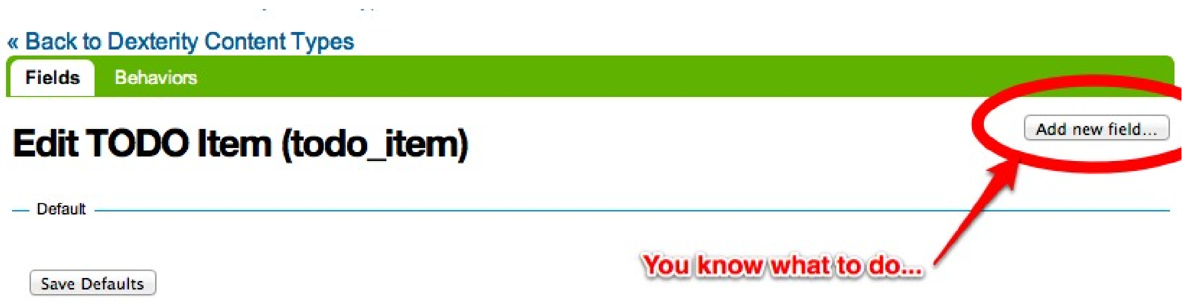
6. By default, all Plone content-types have Dublin Core metadata enabled (you may know it as `title` and `description`. We don't need this for our über simple Todo list item. Uncheck `Dublin Core metadata` and then click `Save`.



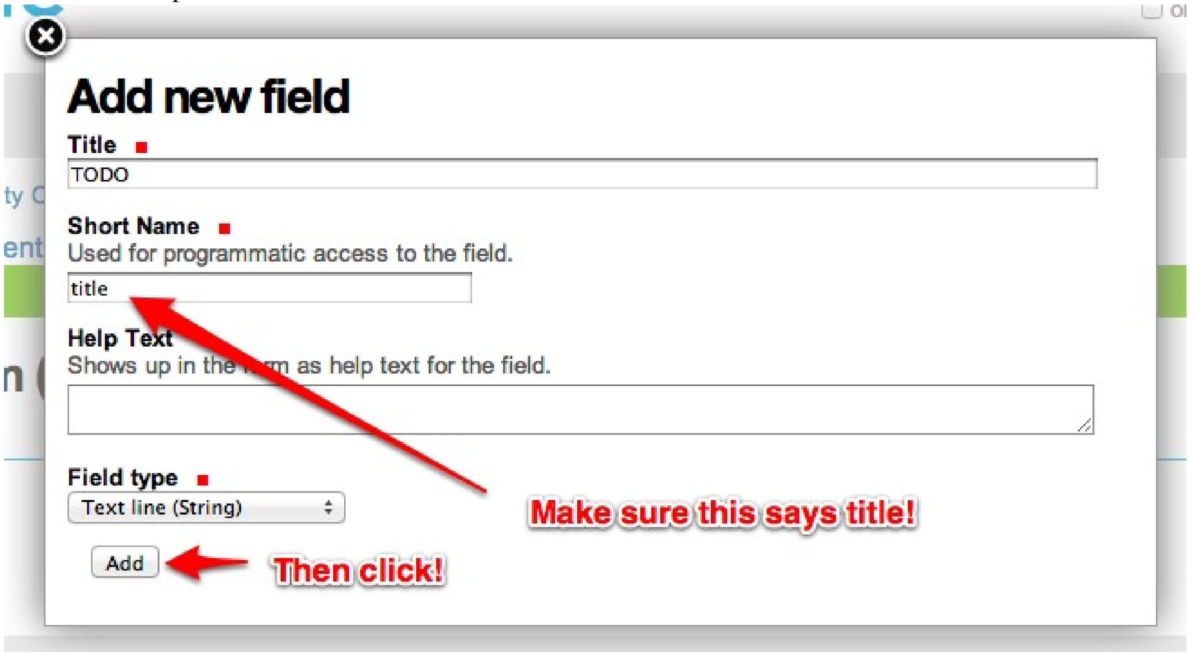
Behaviors for TODO Item (todo_item)

- [ ] Basic metadata
  Adds a title and description
- [ ] Categorization
  Adds keywords and language setting
- [ ] Dublin Core metadata
  Adds Dublin Core metadata fields (equals Basic metadata + Categorization + Effective range + Ownership)
- [ ] Effective range
  Assign an effective and/or expiration date
- [ ] Exclude From navigation
  Allow items to be excluded from navigation
- [ ] Hide rules tab
  Remove the rules tab when viewing this piece of content
- [ ] Hide share tab
  Remove the share tab when viewing this piece of content
- [ ] Name from file name
  Automatically generate short URL name for content based on its primary field file name
- [x] Name from title
  Automatically generate short URL name for content based on its initial title

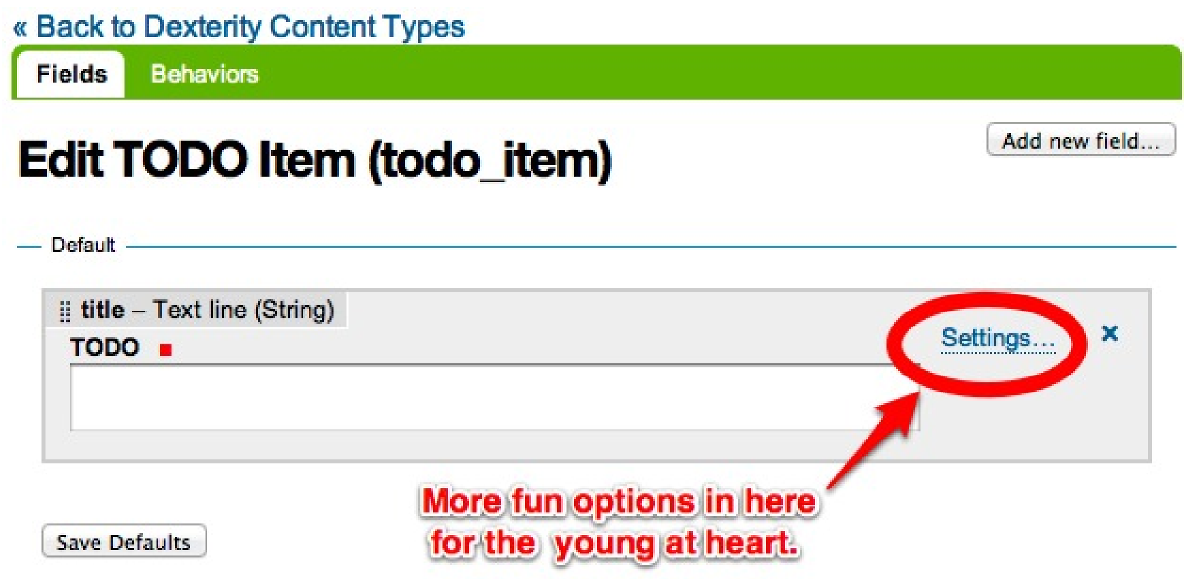
7. Next we need to add some fields. Because this type is so simple, we will add just one field, but feel free to go CRAZY. Start by going back to the `Fields` tab and clicking `Add new field...`.



8. Add a field called `Todo`, or anything else you want. But! Note that it's very important that the *Short Name* field value is `title`. By using this key short name, we make sure that all Todo Items are searchable from smart search. Update the field as seen below and click `Add`.


Add new field
Title
TODO
Short Name
Used for programmatic access to the field.
title
Help Text
Shows up in the form as help text for the field.
Field type
Text line (String)
Add
Make sure this says title!
Then click!

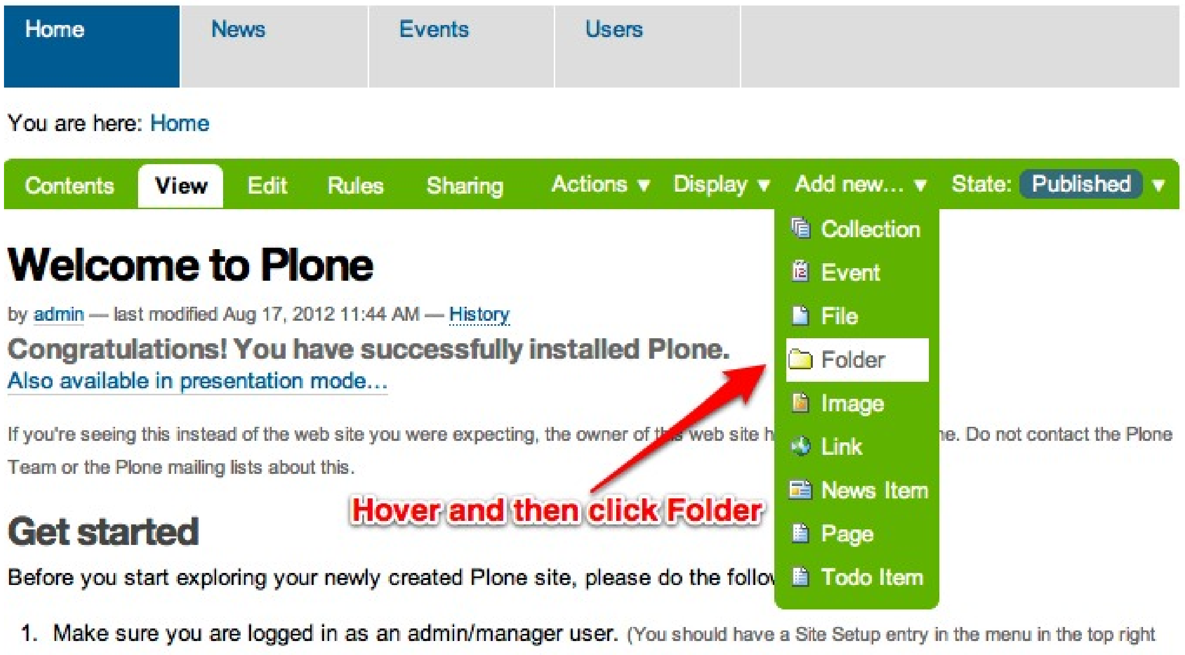
9. You will see that a new field has been added to your content type. If you are feeling adventuresome, click on the settings tab next to the field to set other properties, or just see what's available.


« Back to Dexterity Content Types
Fields
Behaviors
Edit TODO Item (todo_item)
Add new field...
Default
title – Text line (String)
TODO
Settings...
More fun options in here for the young at heart.
Save Defaults

Trying out the Todo Item content-type

Now it's time to reap the rewards of all of your effort. Let's put all of our Todo Items in one particular folder so that we can have collections of items throughout the site. For this tutorial, we will be putting everything in the root of the site so it's easy to debug.

1. From the root, add a new folder called `TODO list`.



Welcome to Plone

by admin — last modified Aug 17, 2012 11:44 AM — History

Congratulations! You have successfully installed Plone.

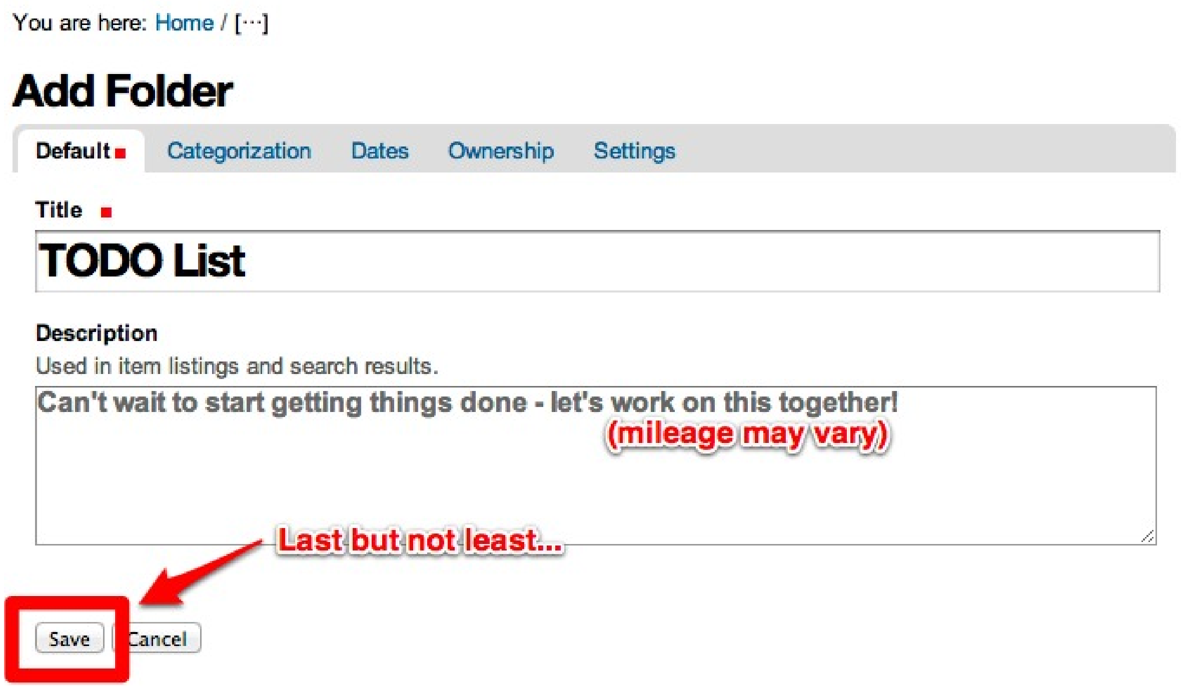
Also available in presentation mode...

If you're seeing this instead of the web site you were expecting, the owner of this web site h... e. Do not contact the Plone Team or the Plone mailing lists about this.

Get started

Before you start exploring your newly created Plone site, please do the follo...

1. Make sure you are logged in as an admin/manager user. (You should have a Site Setup entry in the menu in the top right



You are here: Home / […]

Add Folder

Default ■ Categorization Dates Ownership Settings

Title ■

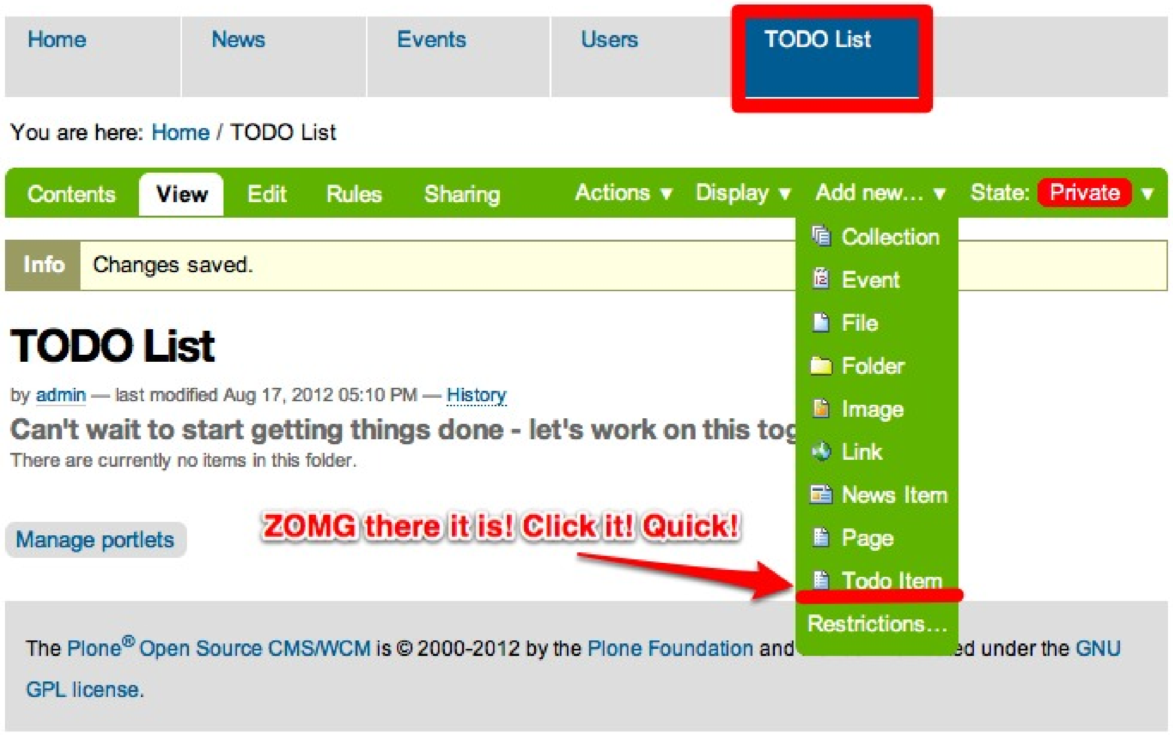
TODO List

Description

Used in item listings and search results.

Can't wait to start getting things done - let's work on this together!

2. Add a new *Todo Item* to the new *Todo* folder.



You are here: Home / TODO List

Info Changes saved.

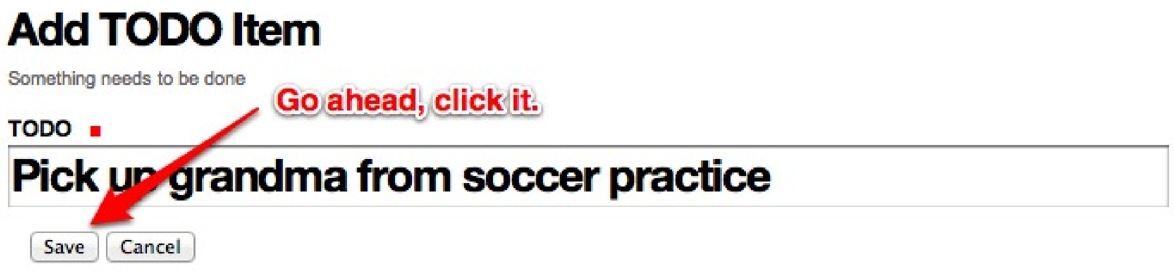
TODO List

by admin — last modified Aug 17, 2012 05:10 PM — History

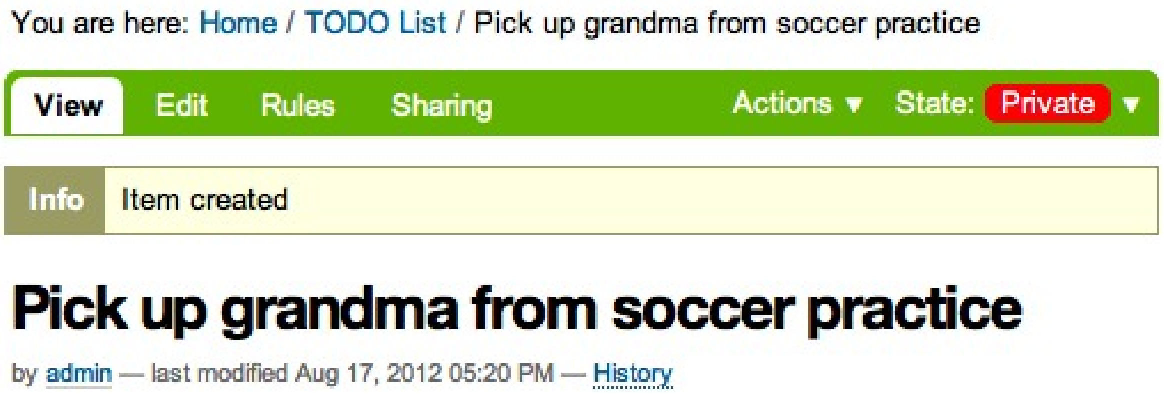
Can't wait to start getting things done - let's work on this tog

There are currently no items in this folder.

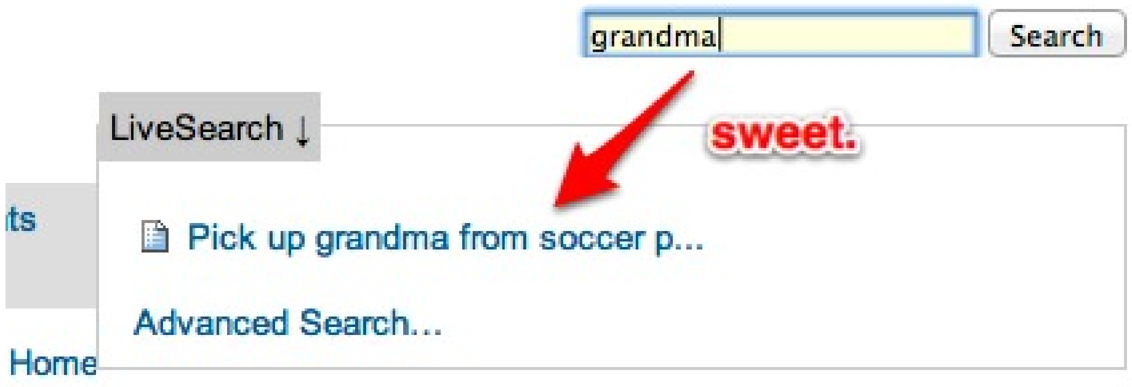
The Plone® Open Source CMS/WCM is © 2000-2012 by the Plone Foundation and …ed under the GNU GPL license.


Add TODO Item
Something needs to be done
Go ahead, click it.
TODO
Pick up grandma from soccer practice
Save
Cancel

3. Celebrate!


You are here: Home / TODO List / Pick up grandma from soccer practice
View
Edit
Rules
Sharing
Actions ▾
State: Private ▾
Info
Item created
Pick up grandma from soccer practice
by admin — last modified Aug 17, 2012 05:20 PM — History

You may be wondering about earlier, when we asked you to make sure that the *short name* for the Todo Item was called `title`. The time has come to let you in on a little secret. Calling the short name either `title` or `description` will automatically add that text to the livesearch menu. WHAT?!? I know! When life gives you lemonade, spike it with vodka and enjoy liberally! You can now search for your Todo Items in Live Search.



But wait a minute... This todo item is marked `private`, and that doesn't really make sense. It's a good thing Plone has an easy solution for that. In the next section, we will go over the basics of that magical, mystical word: *workflow*.

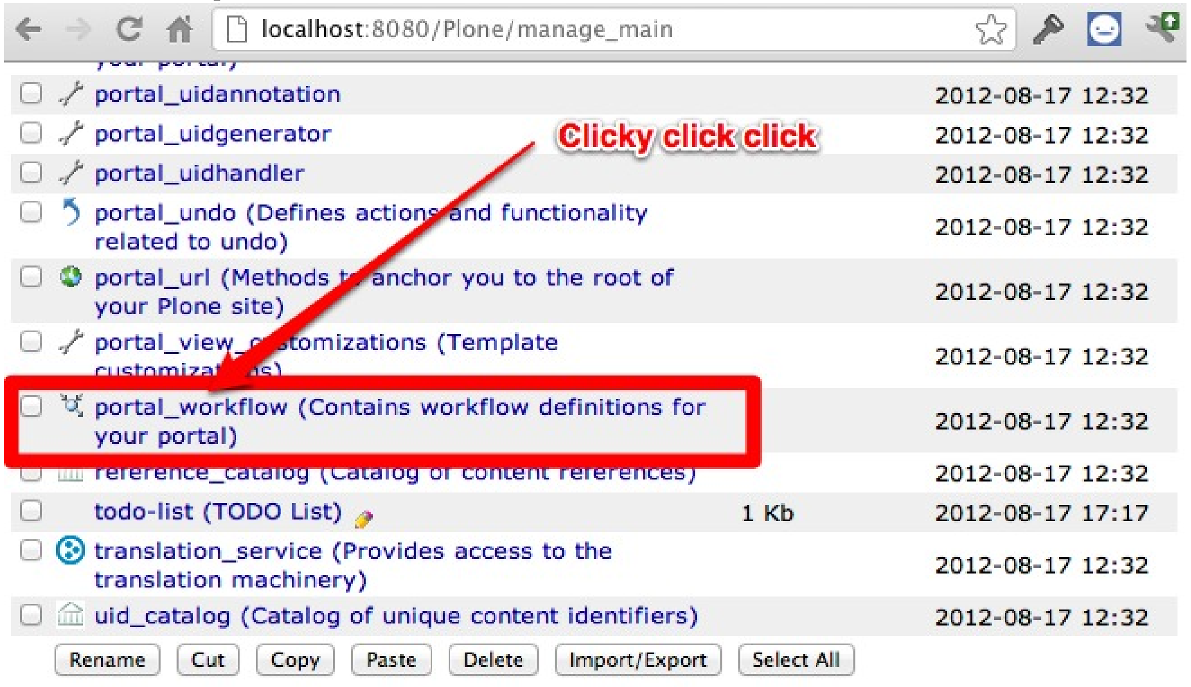
1.5.2 Getting Started with Workflows

So what is a workflow? It is a mechanism to control the flow of a content item through various states in time. Most commonly, and by default in Plone, you deal with a *publication* workflow. For example: A writer writes up a News Item and submits it for review. Then the in-house reviewing team goes through the text and publishes the News Item so it is public for the entire world to see.

The Todo Item we added in the last section is marked as `private` because by default all new Plone content items are assigned a workflow called `simple_publication_workflow`. I know what you are thinking: simple publication whodie whatie grble gobble??!?! Just like before, let's bypass trying to explain what that means and just fix it. Relax, enjoy, and finish the tutorial!

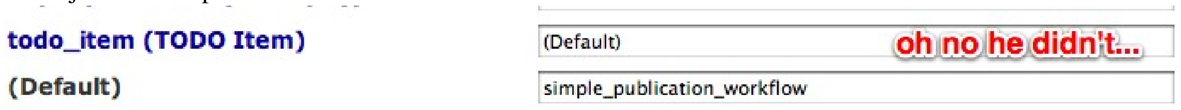
Todo Items really have 2 states that we are interested in: *open* and *complete*. Let's make that happen.

1. Head over to the ZMI at `http://localhost:8080/Plone/manage_main`.
2. In the ZMI, open the `portal_workflow` tool.



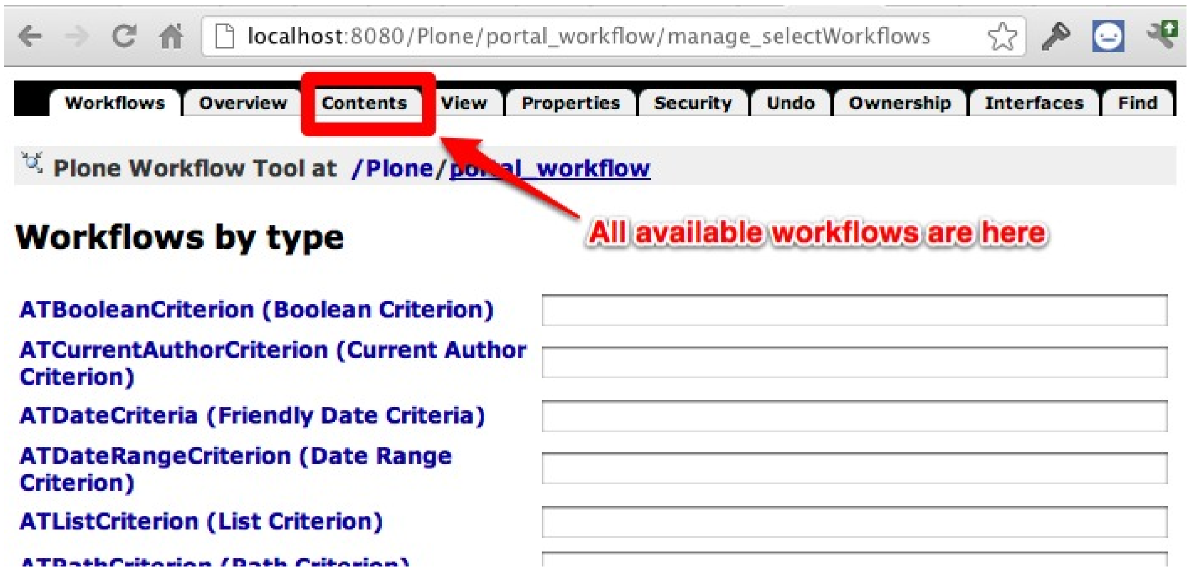
portal_uidannotation		2012-08-17 12:32
portal_uidgenerator		2012-08-17 12:32
portal_uidhandler		2012-08-17 12:32
portal_undo (Defines actions and functionality related to undo)		2012-08-17 12:32
portal_url (Methods to anchor you to the root of your Plone site)		2012-08-17 12:32
portal_view_customizations (Template customizations)		2012-08-17 12:32
portal_workflow (Contains workflow definitions for your portal)		2012-08-17 12:32
reference_catalog (Catalog of content references)		2012-08-17 12:32
todo-list (TODO List)	1 Kb	2012-08-17 17:17
translation_service (Provides access to the translation machinery)		2012-08-17 12:32
uid_catalog (Catalog of unique content identifiers)		2012-08-17 12:32

On this page, we see all content-types in our portal *mapped* to a workflow. Our new type, Todo Item, is mapped to `(Default)`. You can see right below that the default is `Simple Publication Workflow`. This is just too complex for our little Todo Item.



todo_item (TODO Item)	(Default)
(Default)	simple_publication_workflow

3. So let's create a new one that suites our needs perfectly! Click the `contents` tab at the top of the page to get a listing of all the available workflows.


localhost:8080/Plone/portal_workflow/manage_selectWorkflows
Workflows
Overview
Contents
View
Properties
Security
Undo
Ownership
Interfaces
Find
All available workflows are here

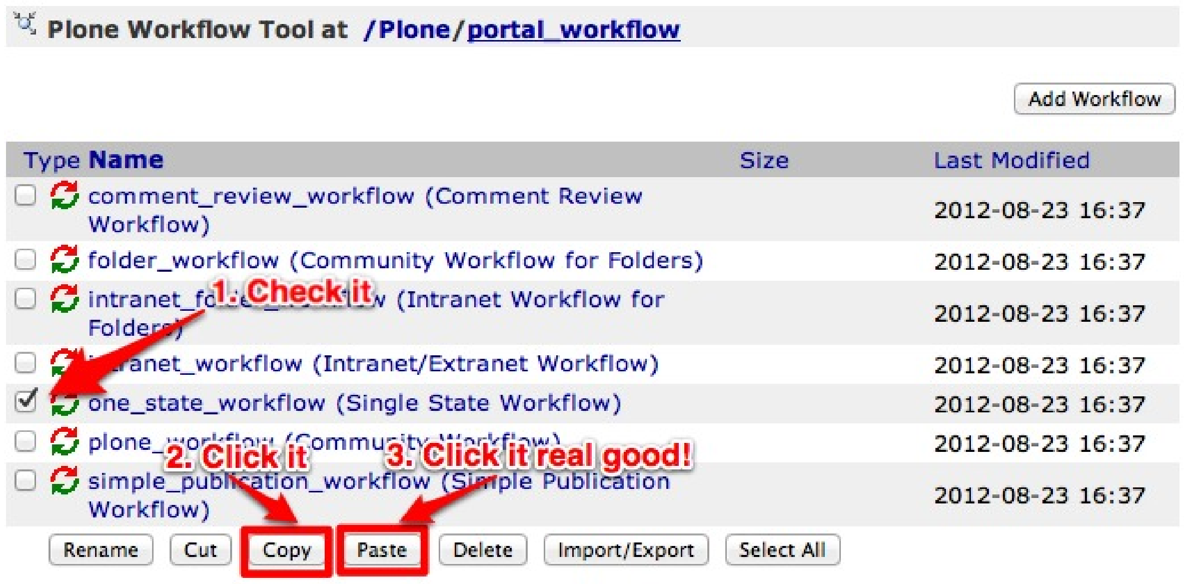
Plone Workflow Tool at /Plone/portal_workflow

Workflows by type

ATBooleanCriterion (Boolean Criterion)
ATCurrentAuthorCriterion (Current Author Criterion)
ATDateCriteria (Friendly Date Criteria)
ATDateRangeCriterion (Date Range Criterion)
ATListCriterion (List Criterion)

You can poke around here all you like, but the details of each one of these workflows are better left to another tutorial. When in doubt, you can always come back to these workflows to see examples of how things can be done. Onwards and upwards!

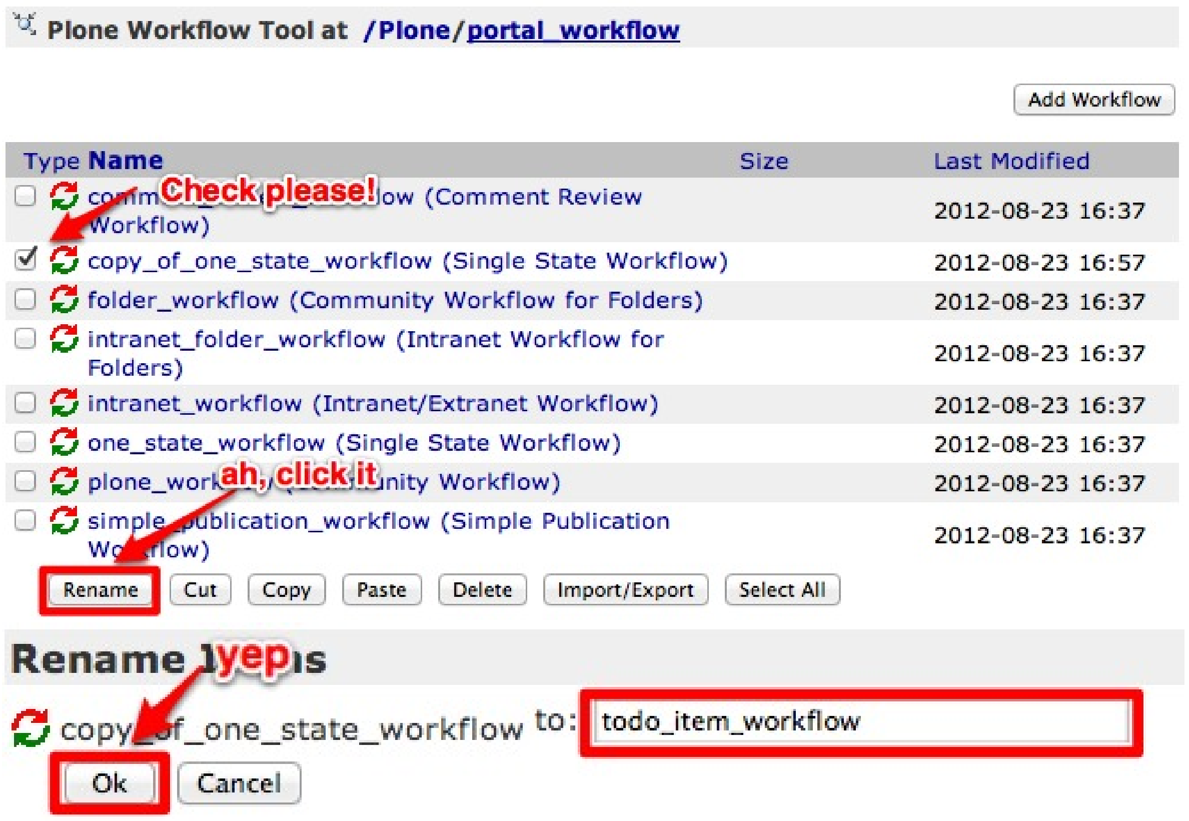
4. Let's create a new workflow for our Todo Items and call it `todo_item_workflow`. We will make a new workflow by copying and customising one of the workflows that are already there. Duplicate the `one_state_workflow`.



Plone Workflow Tool at /Plone/portal_workflow

	Type	Name	Size	Last Modified
☐		comment_review_workflow (Comment Review Workflow)		2012-08-23 16:37
☐		folder_workflow (Community Workflow for Folders)		2012-08-23 16:37
☐		intranet_folder_workflow (Intranet Workflow for Folders)		2012-08-23 16:37
☐		intranet_workflow (Intranet/Extranet Workflow)		2012-08-23 16:37
☑		one_state_workflow (Single State Workflow)		2012-08-23 16:37
☐		plone_workflow (Community Workflow)		2012-08-23 16:37
☐		simple_publication_workflow (Simple Publication Workflow)		2012-08-23 16:37

5. Rename the copied workflow to `todo_item_workflow`.



Plone Workflow Tool at /Plone/portal_workflow

Add Workflow

Type	Name	Size	Last Modified
☐	comment_review_workflow (Comment Review Workflow)		2012-08-23 16:37
✓	copy_of_one_state_workflow (Single State Workflow)		2012-08-23 16:57
☐	folder_workflow (Community Workflow for Folders)		2012-08-23 16:37
☐	intranet_folder_workflow (Intranet Workflow for Folders)		2012-08-23 16:37
☐	intranet_workflow (Intranet/Extranet Workflow)		2012-08-23 16:37
☐	one_state_workflow (Single State Workflow)		2012-08-23 16:37
☐	plone_workflow (Community Workflow)		2012-08-23 16:37
☐	simple_publication_workflow (Simple Publication Workflow)		2012-08-23 16:37

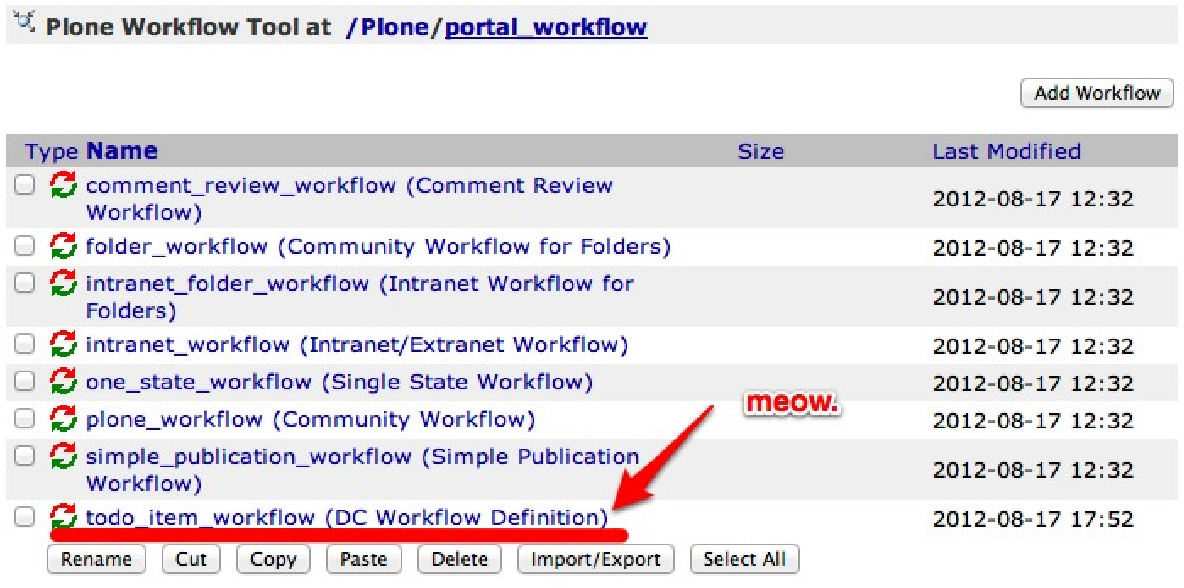
Rename | Cut | Copy | Paste | Delete | Import/Export | Select All

Rename Items

copy_of_one_state_workflow to: todo_item_workflow

Ok | Cancel

6. You will be spit back out to the workflow contents page. Click the workflow to start editing.



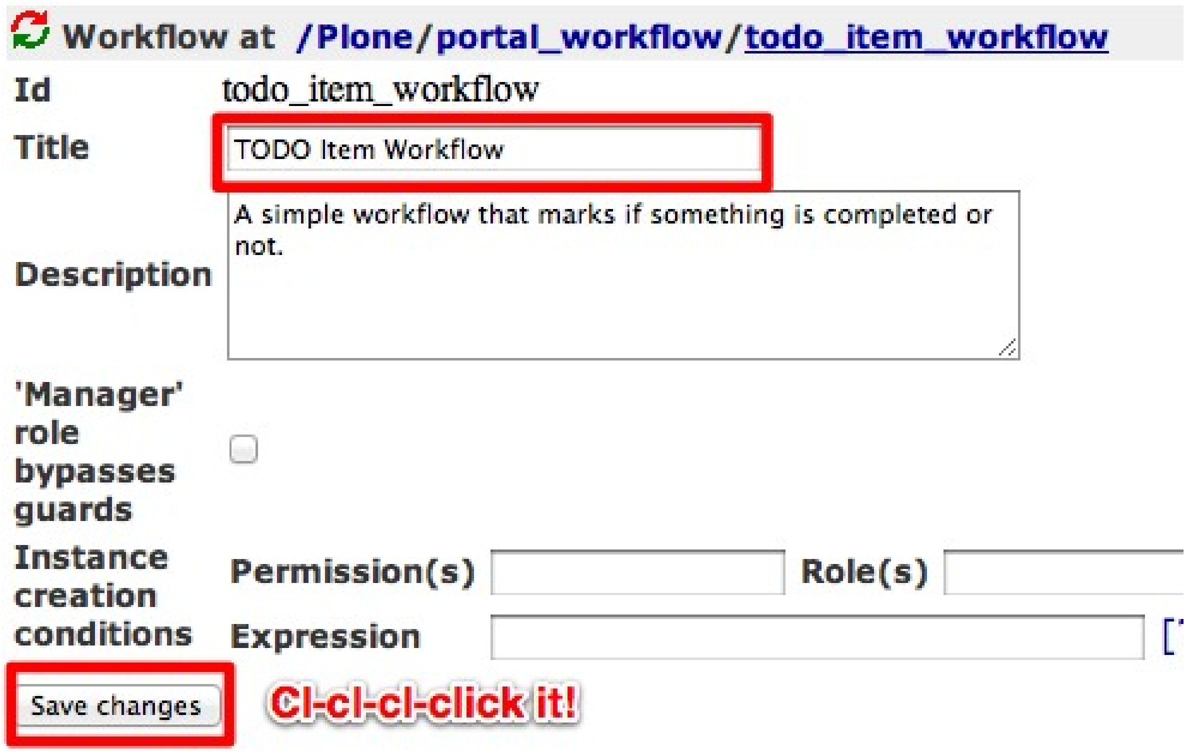
Plone Workflow Tool at /Plone/portal_workflow

Add Workflow

Type	Name	Size	Last Modified
☐	comment_review_workflow (Comment Review Workflow)		2012-08-17 12:32
☐	folder_workflow (Community Workflow for Folders)		2012-08-17 12:32
☐	intranet_folder_workflow (Intranet Workflow for Folders)		2012-08-17 12:32
☐	intranet_workflow (Intranet/Extranet Workflow)		2012-08-17 12:32
☐	one_state_workflow (Single State Workflow)		2012-08-17 12:32
☐	plone_workflow (Community Workflow)		2012-08-17 12:32
☐	simple_publication_workflow (Simple Publication Workflow)		2012-08-17 12:32
☐	todo_item_workflow (DC Workflow Definition)		2012-08-17 17:52

Rename | Cut | Copy | Paste | Delete | Import/Export | Select All

7. Let’s update the name of the workflow so we don’t double take later on.


Workflow at /Plone/portal_workflow/todo_item_workflow

Id todo_item_workflow

Title: TODO Item Workflow

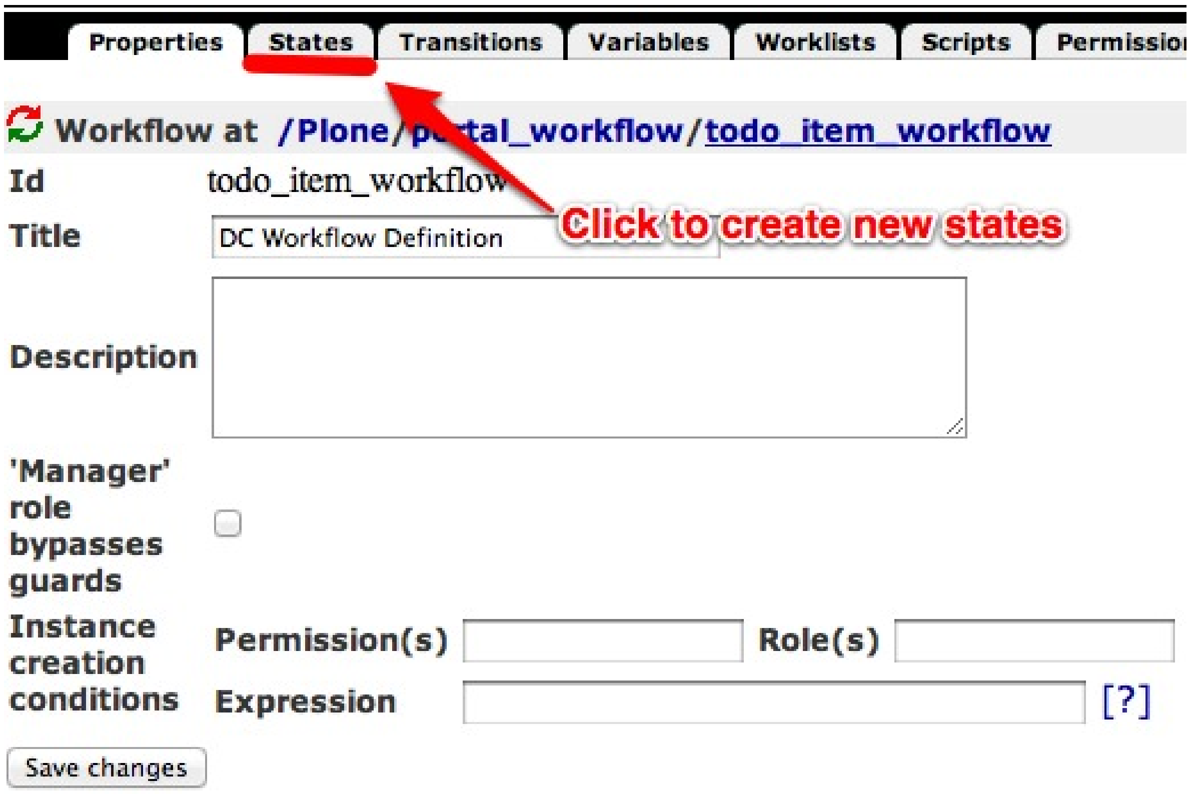
Description: A simple workflow that marks if something is completed or not.

'Manager' role bypasses guards: ☐

Instance creation conditions: Permission(s) Role(s) Expression

8. Workflow is something that takes time to get used to if you have never encountered the concept. The best analogy in our case is to a car. The car engine has two simple states: *on* and *off*. To transition from on to off and vice versa, it needs some action from the driver. The same for our TODO items. They have two states: *open* and *completed*. In order to get them from *open* to *completed*, the user needs to click something. Don't understand yet? Relax, sit back, and finish the tutorial.

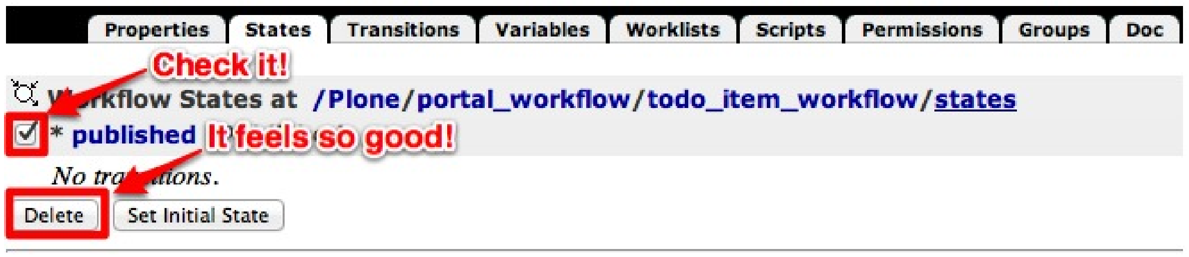
   Lets start by adding our base states. We will call them *open* and *complete*. From the edit workflow screen, click on the `States` tab.



Workflow at /Plone/portal_workflow/todo_item_workflow

Id	todo_item_workflow
Title	DC Workflow Definition
Description	
'Manager' role bypasses guards	☐
Instance creation conditions	Permission(s) Role(s) Expression

9. Delete the currently listed state.

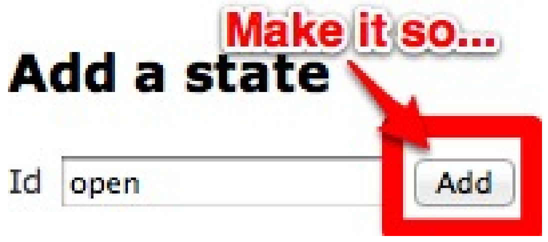


Workflow States at /Plone/portal_workflow/todo_item_workflow/states

☑ * published

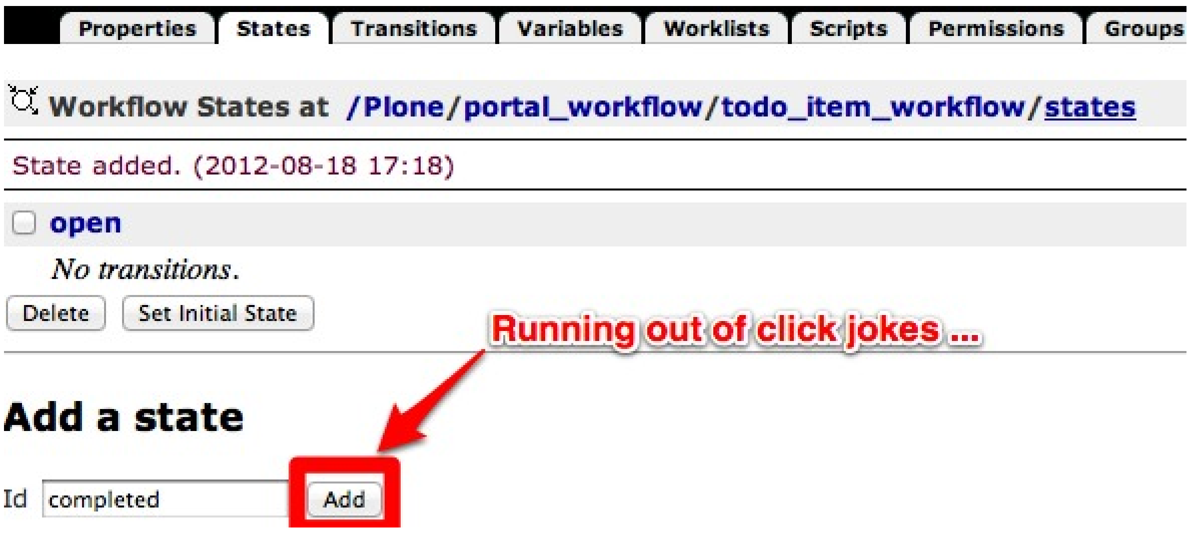
No transitions.

10. Add two states with the ids `open` and `completed`.



Add a state

Id open



Workflow States at /Plone/portal_workflow/todo_item_workflow/states

State added. (2012-08-18 17:18)

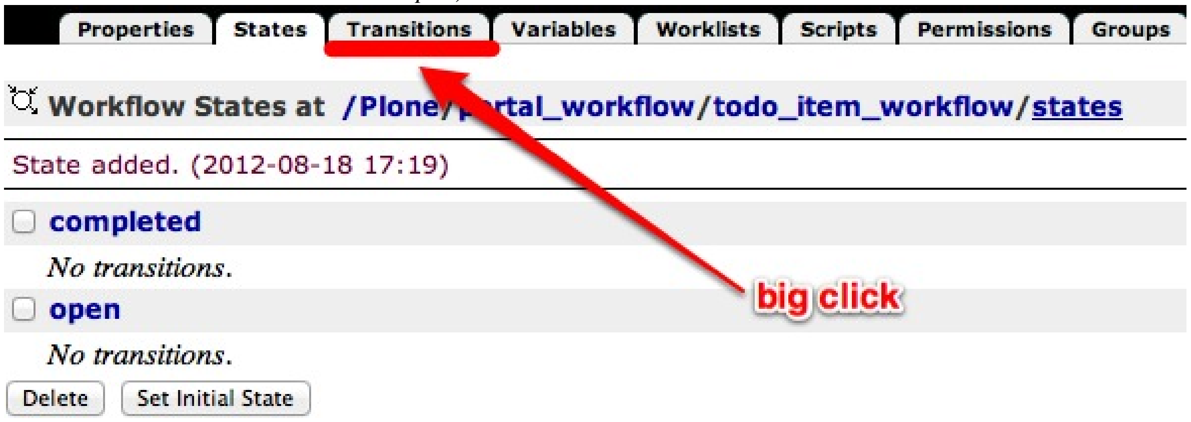
open

No transitions.

Add a state

Id completed

11. Next lets add *transitions*. They will take the TODO item from *open* to *completed* and vice versa (in case a user wants to revert an item back to *open*). Click on the `Transitions` tab.



Workflow States at /Plone/portal_workflow/todo_item_workflow/states

State added. (2012-08-18 17:19)

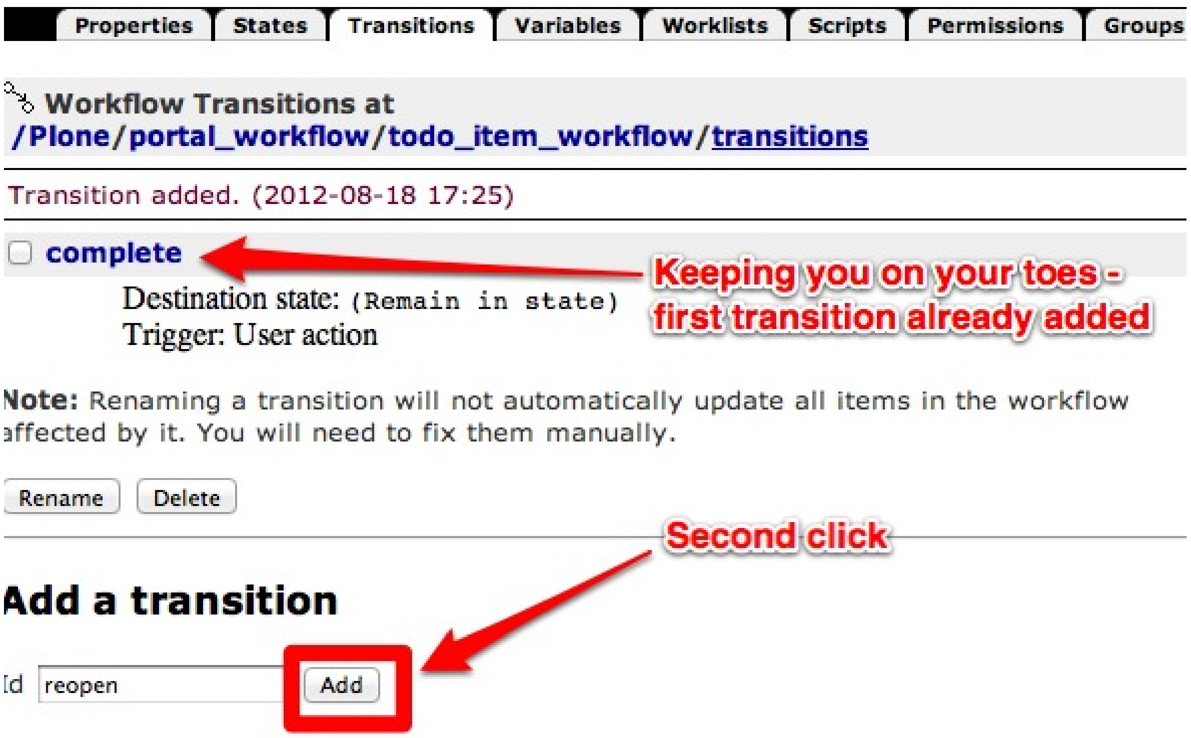
completed

No transitions.

open

No transitions.

12. Add two transitions: *complete* and *reopen*. When a user *completes* a task, it will move into the *completed* state. When a user *reopens* a task, it will go back to the *open* state.



Workflow Transitions at /Plone/portal_workflow/todo_item_workflow/transitions

Transition added. (2012-08-18 17:25)

complete

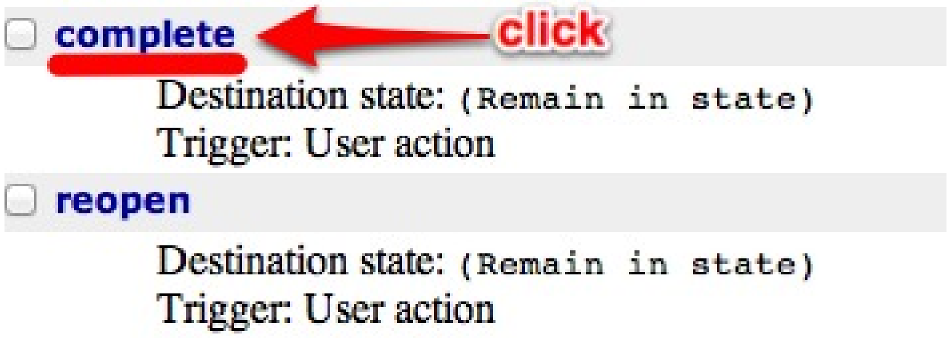
Destination state: (Remain in state)
Trigger: User action

Note: Renaming a transition will not automatically update all items in the workflow affected by it. You will need to fix them manually.

Add a transition

Id reopen

13. Let's add a few details to these new transitions. Let's start with *complete*. Click on `complete` to edit the transition.



complete

Destination state: (Remain in state)
Trigger: User action

reopen

Destination state: (Remain in state)
Trigger: User action

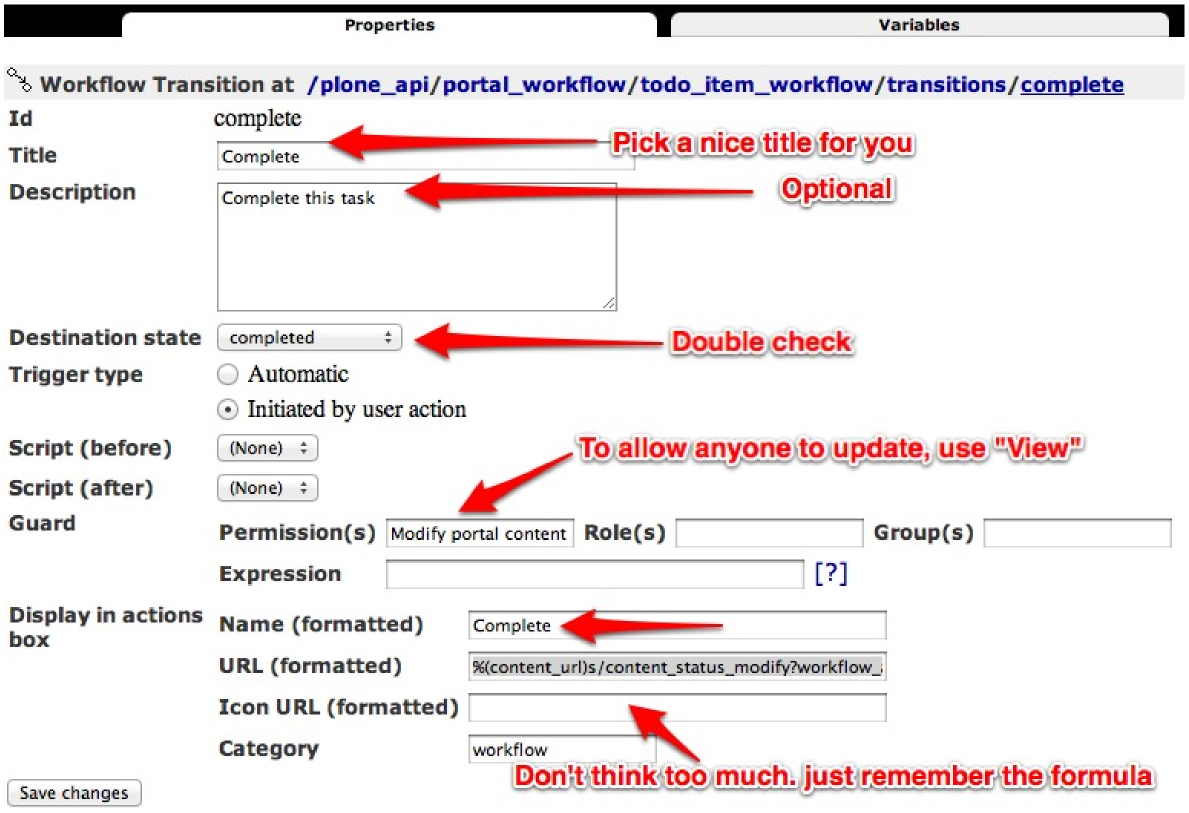
14. First add a title so you remember later what this does. Description is optional but adding one will help you keep your thoughts clear and remind the *future you* what the *today you* is thinking. The destination state should be set to `completed`. We also want to make sure that only people with mega permissions, or the creator of the todo item itself, can change the state so we add `Modify portal content` to the `Permissions` box.

    All this means nothing if we don't give the user a chance to change the state. Next to `Display in actions box`, we can set the title for what will be displayed in the workflow drop down box of the item (where *Pending*, *Reject*, etc. where earlier). Let's call it `Complete`. Last but not least, we need to add the URL that the action points to. I could make this tutorial 100 years long and explain why you have to do this, but accept that it has to be done, relax, and follow this formula:

    ```
    URL = %(content_url)s/content_status_modify?workflow_action=X
    ```

    where *X* is the id of the transition. So for this case, in the URL box, you will add

```
%(content_url)s/content_status_modify?workflow_action=complete
```



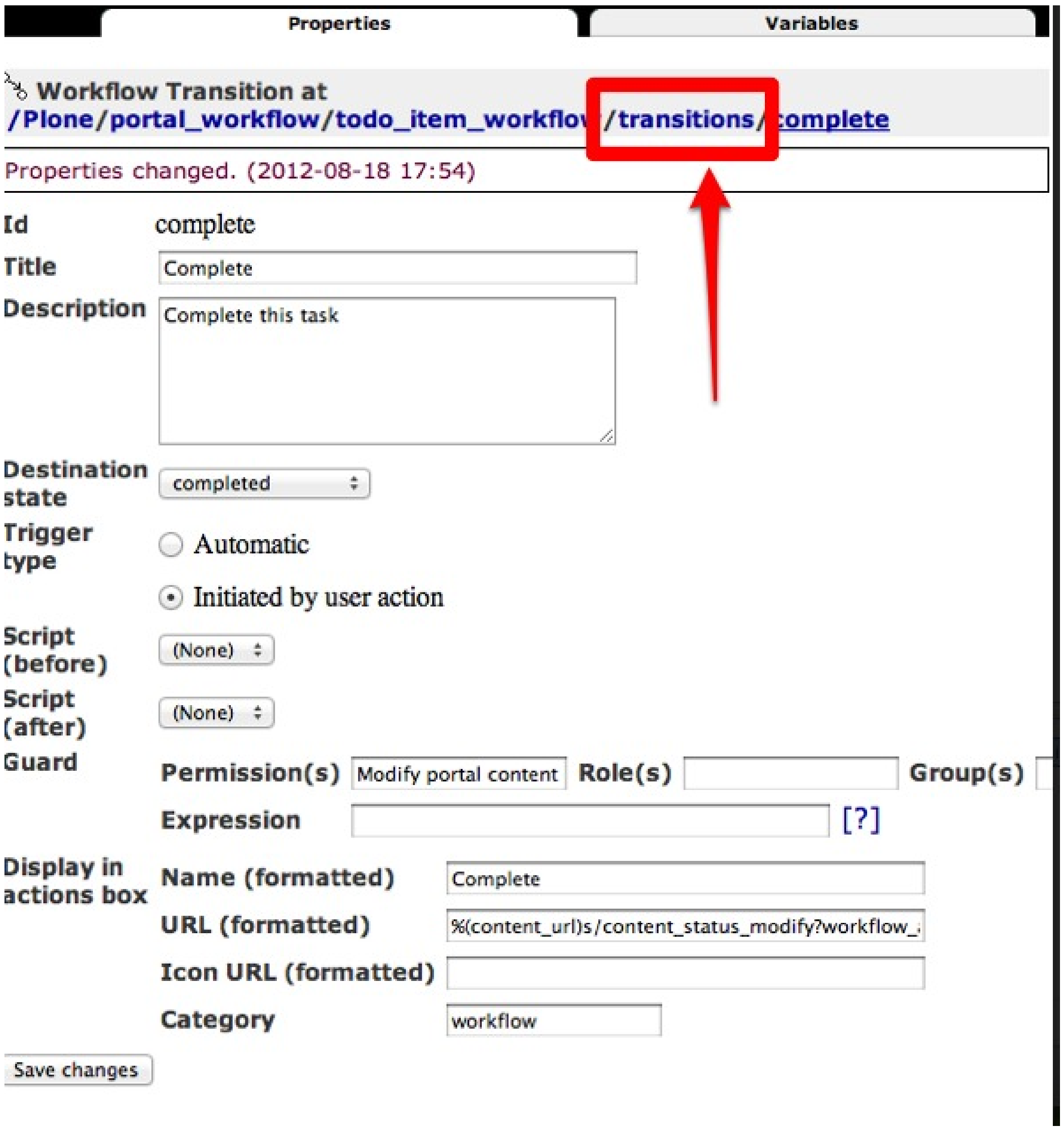
Workflow Transition at /plone_api/portal_workflow/todo_item_workflow/transitions/complete

Id	complete
Title	Complete
Description	Complete this task
Destination state	completed
Trigger type	( ) Automatic (•) Initiated by user action
Script (before)	(None)
Script (after)	(None)
Guard	Permission(s): Modify portal content; Role(s): ; Group(s): Expression: [?]
Display in actions box	Name (formatted): Complete URL (formatted): %(content_url)s/content_status_modify?workflow_ Icon URL (formatted): Category: workflow

Pick a nice title for you
Optional
Double check
To allow anyone to update, use "View"
Don't think too much. just remember the formula

Double check everything and click `Save`.

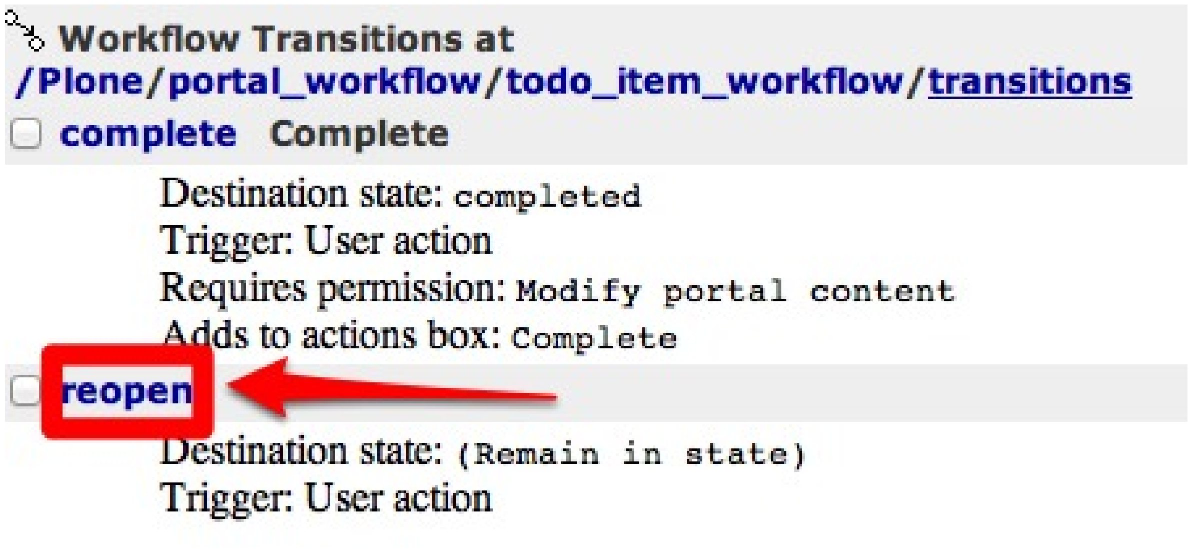
15. If your brain isn't hurting yet it will be soon. Go back to the transitions listing.



Workflow Transition at /Plone/portal_workflow/todo_item_workflow/transitions/complete

Properties changed. (2012-08-18 17:54)

Id	complete
Title	Complete
Description	Complete this task
Destination state	completed
Trigger type	( ) Automatic [✓ Initiated by user action]
Script (before)	(None)
Script (after)	(None)
Guard	Permission(s): Modify portal content; Role(s): ; Group(s): ; Expression:
Display in actions box	Name (formatted): Complete; URL (formatted): %(content_url)s/content_status_modify?workflow_; Icon URL (formatted): ; Category: workflow



Workflow Transitions at /Plone/portal_workflow/todo_item_workflow/transitions

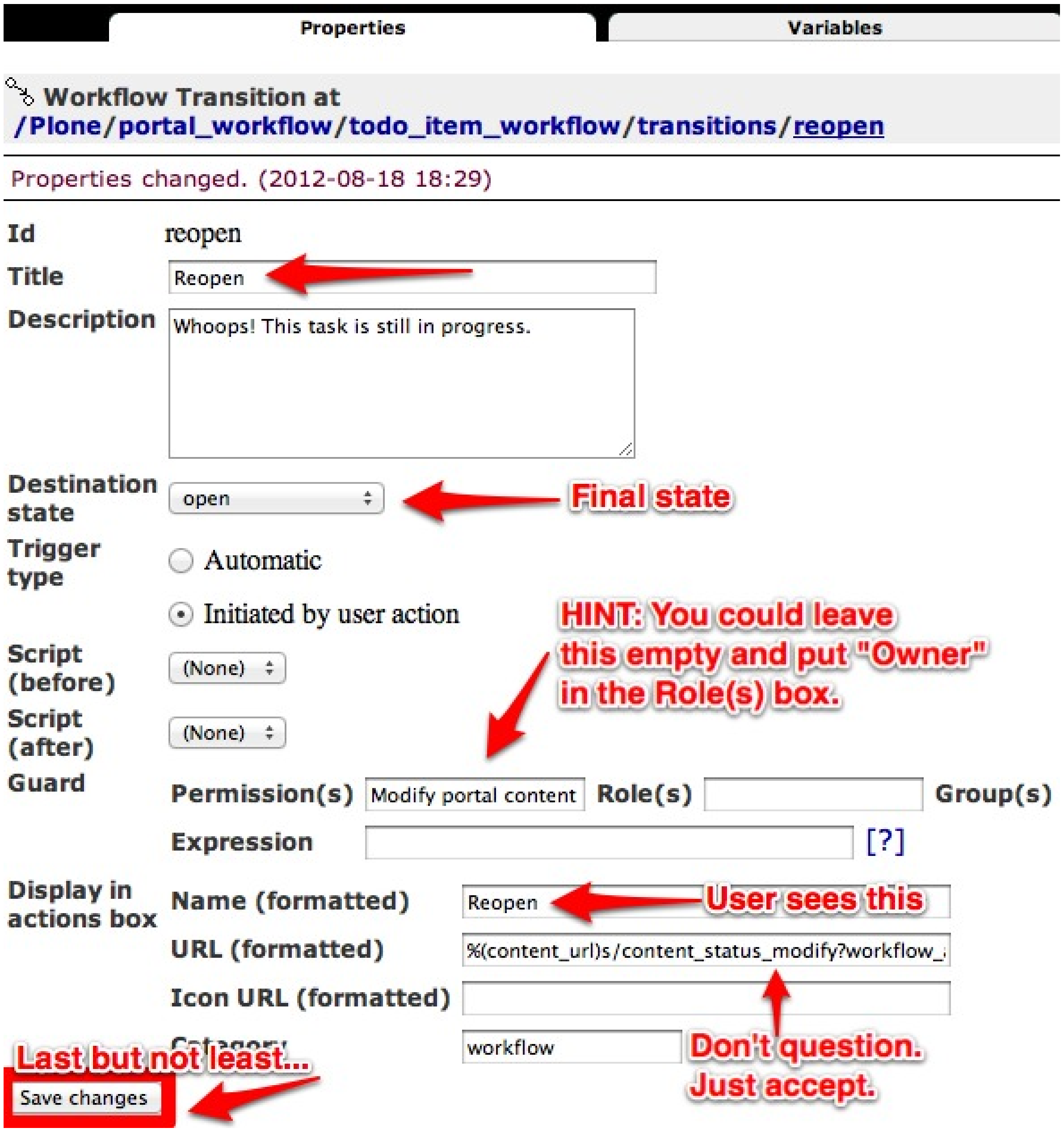
complete Complete

Destination state: `completed`
Trigger: User action
Requires permission: `Modify portal content`
Adds to actions box: `Complete`

reopen

Destination state: `(Remain in state)`
Trigger: User action

16. Let's update the *reopen* transition and update in a similar manner. This time, the destination state is `open`, and following the formula above, the URL is `%(content_url)s/content_status_modify?workflow_action=reopen`.



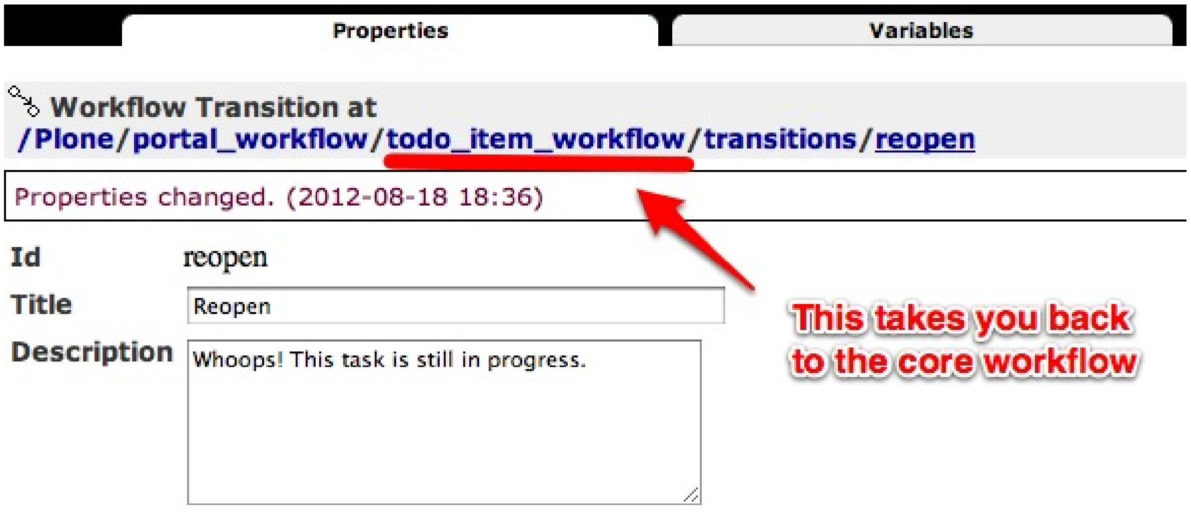
Workflow Transition at /Plone/portal_workflow/todo_item_workflow/transitions/reopen

Properties changed. (2012-08-18 18:29)

Id	reopen
Title	Reopen
Description	Whoops! This task is still in progress.
Destination state	open
Trigger type	( ) Automatic [✓ Initiated by user action]
Script (before)	(None)
Script (after)	(None)
Guard	Permission(s): Modify portal content; Role(s): ; Group(s): ; Expression: [?]
Display in actions box	Name (formatted): Reopen; URL (formatted): %(content_url)s/content_status_modify?workflow_; Icon URL (formatted): ; Category: workflow

HINT: You could leave this empty and put "Owner" in the Role(s) box.

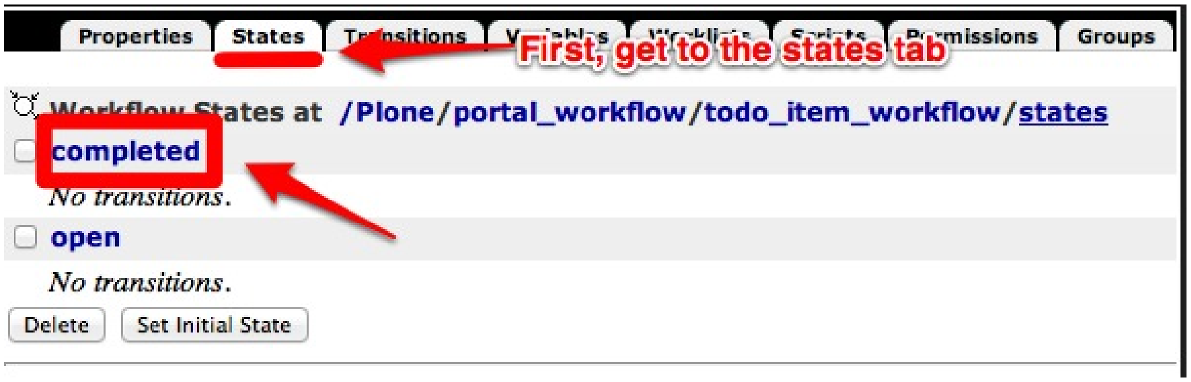
17. Now we have 2 states and 2 transitions, but they aren't 100% linked together ... yet. Go back to the workflow listing, click the `States` tab and then click on `completed` to edit the state.



Workflow Transition at /Plone/portal_workflow/todo_item_workflow/transitions/reopen

Properties changed. (2012-08-18 18:36)

Id	reopen
Title	Reopen
Description	Whoops! This task is still in progress.

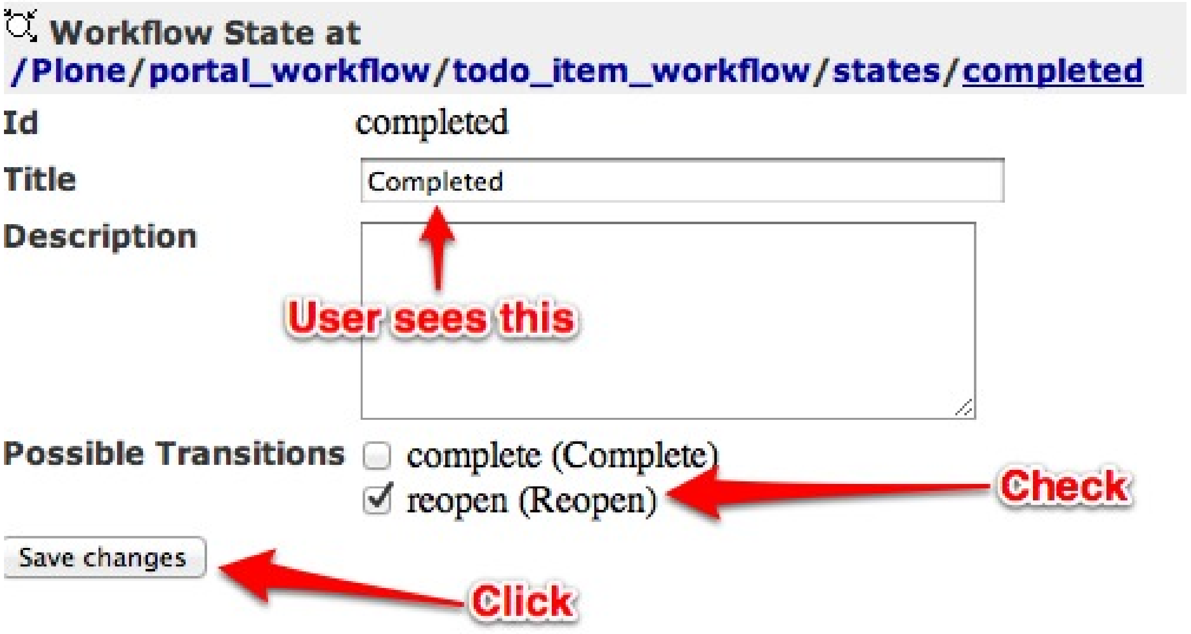


Workflow States at /Plone/portal_workflow/todo_item_workflow/states

completed
No transitions.

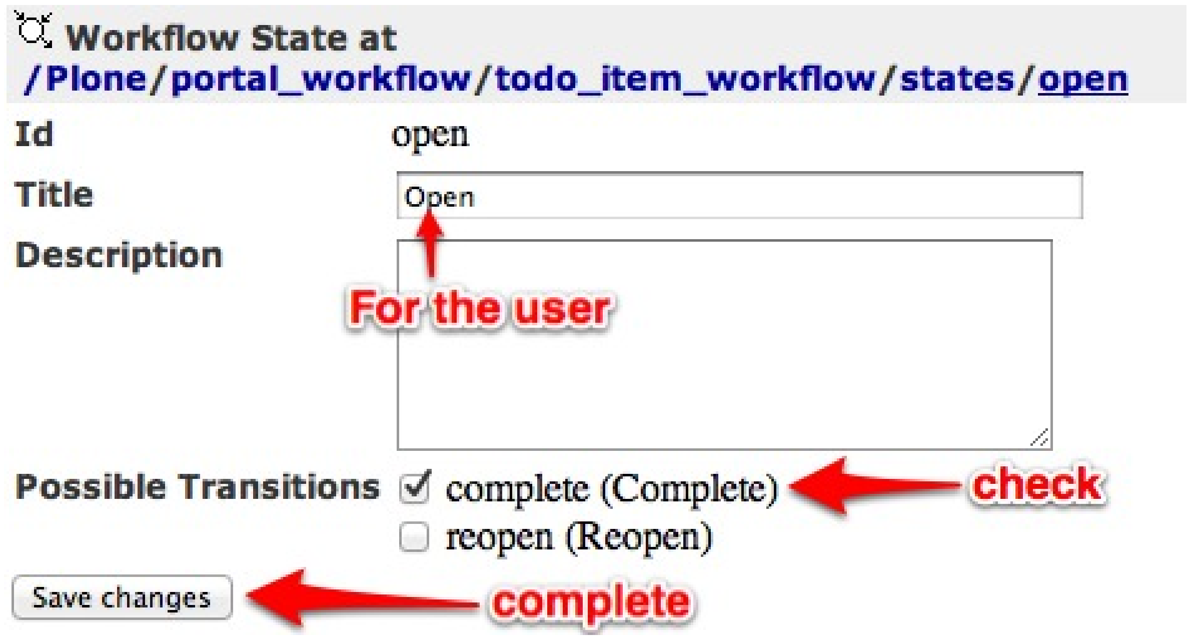
open
No transitions.

18. Add a title, since this is what users see in the top right corner of the TODO items, and then check `reopen` as a possible transition. This means that when a TODO item is completed, it will only allow the user to reopen it (and not re-complete it, for example). In the same respect, open the `open` transition, add a title, and mark `complete` as a possible transition.



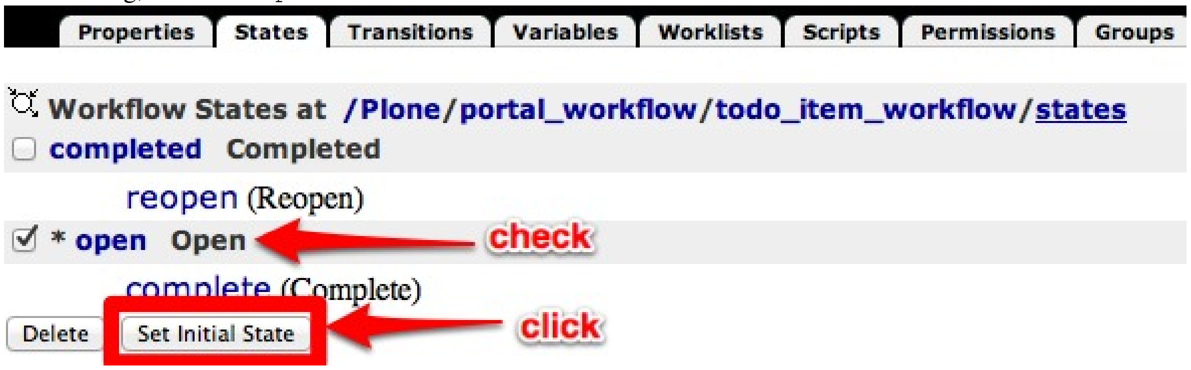
Workflow State at /Plone/portal_workflow/todo_item_workflow/states/completed

Id	completed
Title	Completed
Description	
Possible Transitions	☐ complete (Complete) ☑ reopen (Reopen)


Workflow State at /Plone/portal_workflow/todo_item_workflow/states/open

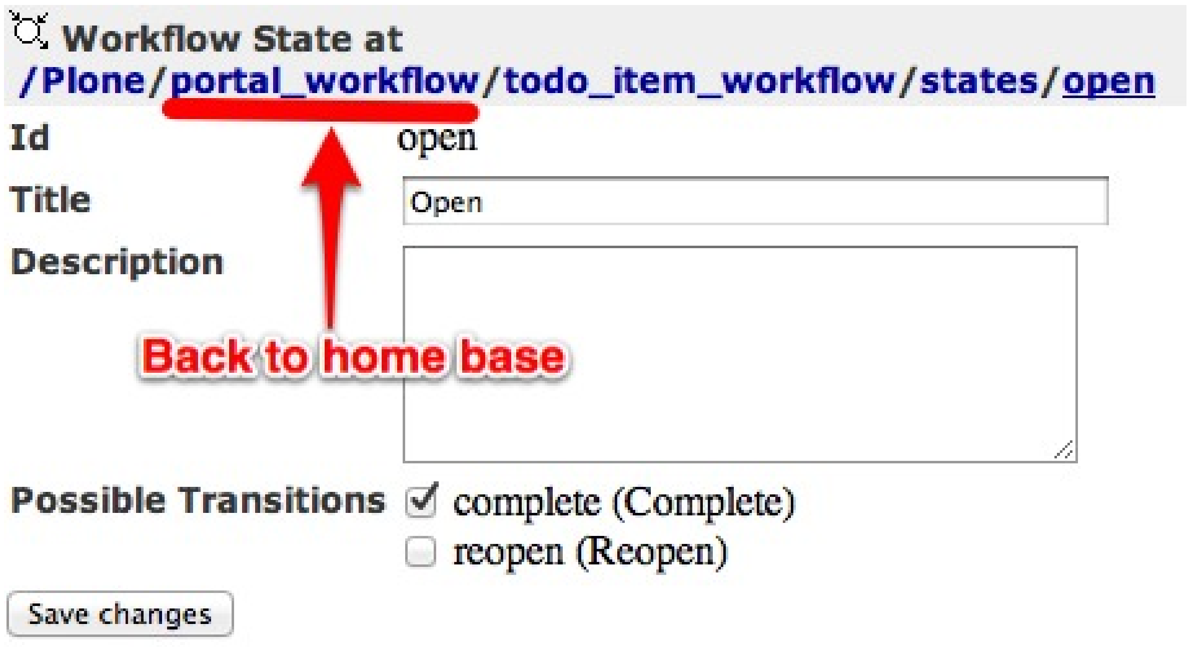
Id	open
Title	Open
Description	
Possible Transitions	✓ complete (Complete)
	☐ reopen (Reopen)

19. When we create a new TODO item, we need to tell Plone what the first state is. Go back to the workflow states listing, and make `open` the initial state.


Workflow States at /Plone/portal_workflow/todo_item_workflow/states

☐ completed Completed
reopen (Reopen)
✓ * open Open
complete (Complete)

20. And that's it! Almost... Last but not least, we need to assign our new workflow to our TODO item type. Go back to the main workflow screen.

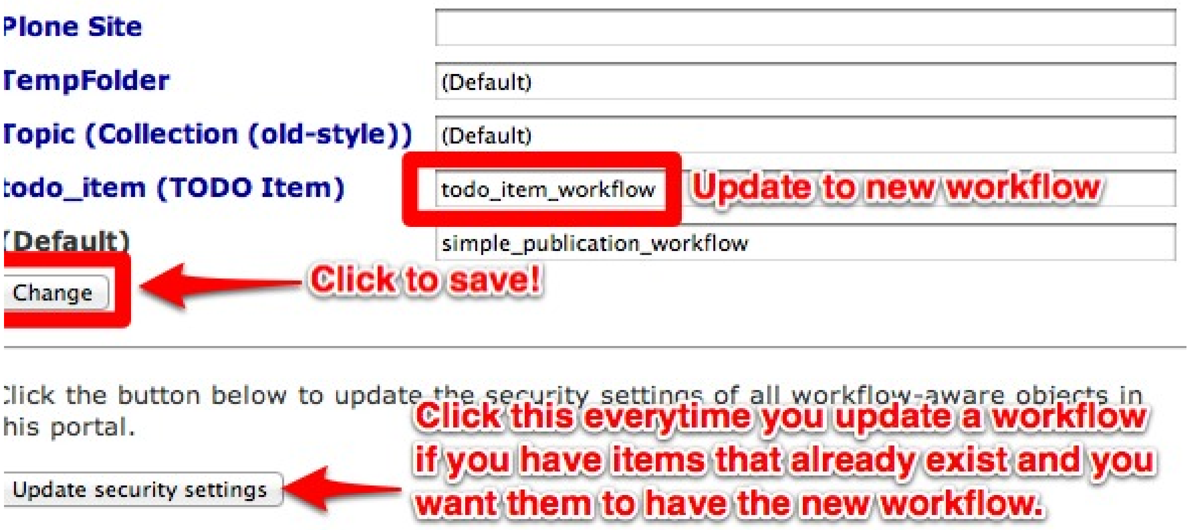


Workflow State at /Plone/portal_workflow/todo_item_workflow/states/open

Id	open
Title	Open
Description	
Possible Transitions	[✓] complete (Complete)
	[ ] reopen (Reopen)

21. Instead of mapping to the `(Default)` workflow, we are going to map to the id of our new workflow, `todo_item_workflow`, and then click `Change`.

    If you already have TODO items in your site, you MUST click `Update Security Settings` to update the workflow for the items. Instead of going into gross detail about why this is the case, just sit back, relax, finish the tutorial, and remember to click this button any time you make changes (yes! you can continue to change and update your workflows!).



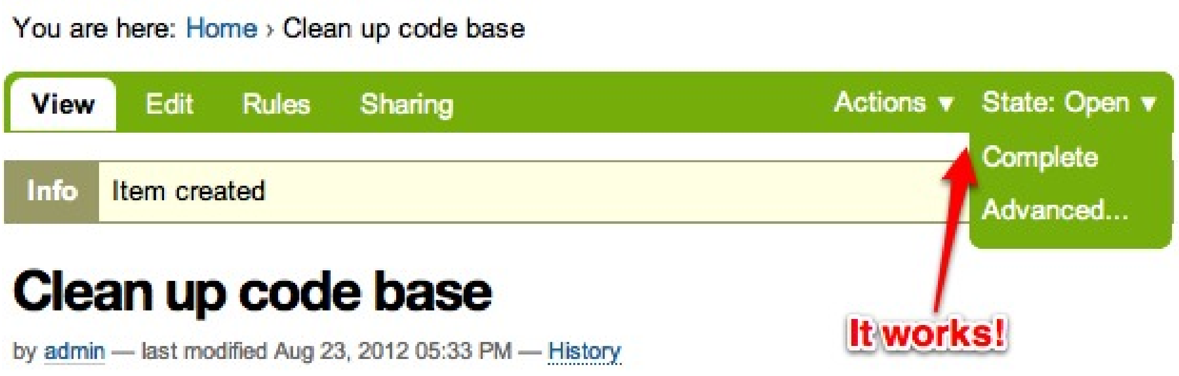
Plone Site	
TempFolder	(Default)
Topic (Collection (old-style))	(Default)
todo_item (TODO Item)	todo_item_workflow
(Default)	simple_publication_workflow

Update to new workflow
Change
Click to save!

Click the button below to update the security settings of all workflow-aware objects in this portal.

Update security settings
Click this everytime you update a workflow if you have items that already exist and you want them to have the new workflow.

22. Could the time have arrived? Time to try it out? YES! Go to your `Todo` folder and add a new TODO Item. Validate that the workflow works as expected. By toggling between the states.



Clean up code base

by admin — last modified Aug 23, 2012 05:33 PM — History

Congrats! You have now passed *Plone Workflow 101*. Next we will transition from developing through the web (TTW) to developing on the filesystem.

WARNING: If you are reading this on GitHub, DON'T! Read it on ReadTheDocs:
http://tutorialtodoapp.readthedocs.org/en/latest/chapter_2.html so you
have working references and proper formatting.

1.6 Chapter 2: Filesystem package

Alright! In Chapter 1 you got your content-type and your workflow hooked up and running. You're now ready for the next step: pushing your changes to a filesystem-based package and into a version control system.

Now, why would you even want to do that? Here's a couple of reasons:

1. **Tracking of changes**

   The most obvious one: when you store the configuration of your content-type and your workflow in a VCS, you can track how they changed over time. It's useful to be able to look back a few months and see how your files changed.

2. **Distribution to other developers**

   If you are working in a team you have two ways of distributing your work: either write up a guide on what needs to be clicked for someone to come to the state you are currently at (slow, manual and error-prone) OR you export your configuration and the other developer simply imports it (fast, consistent).

3. **Tests**

   Last, but the most important one, having your configuration exported to a filesystem package allows you to write tests for it. When your test runner spins up a Plone site to run tests against, it needs to have the same content-type and workflow that you configured TTW. And importing configuration is by far the easiest way to give the test runner just that.

1.6.1 Package skeleton

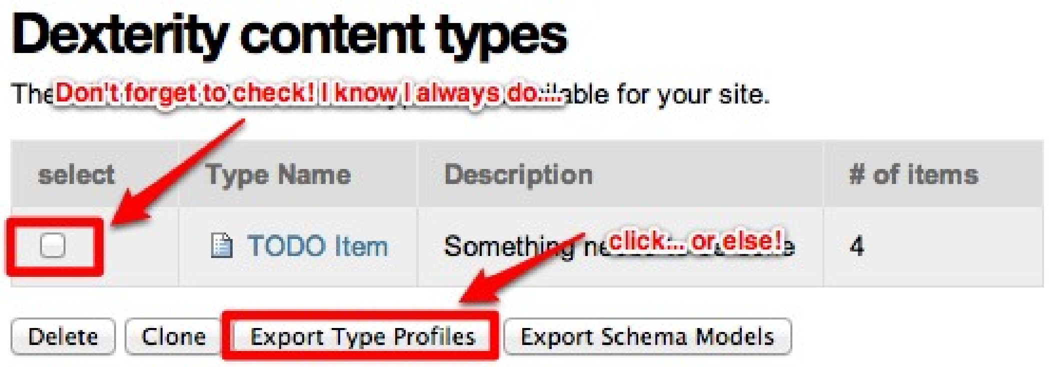
Let's start by creating a package skeleton. Since writing things up from scratch kinda sucks, use this tutorial as your skeleton.

1.6.2 Exporting configuration

Exporting Todo Item content-type

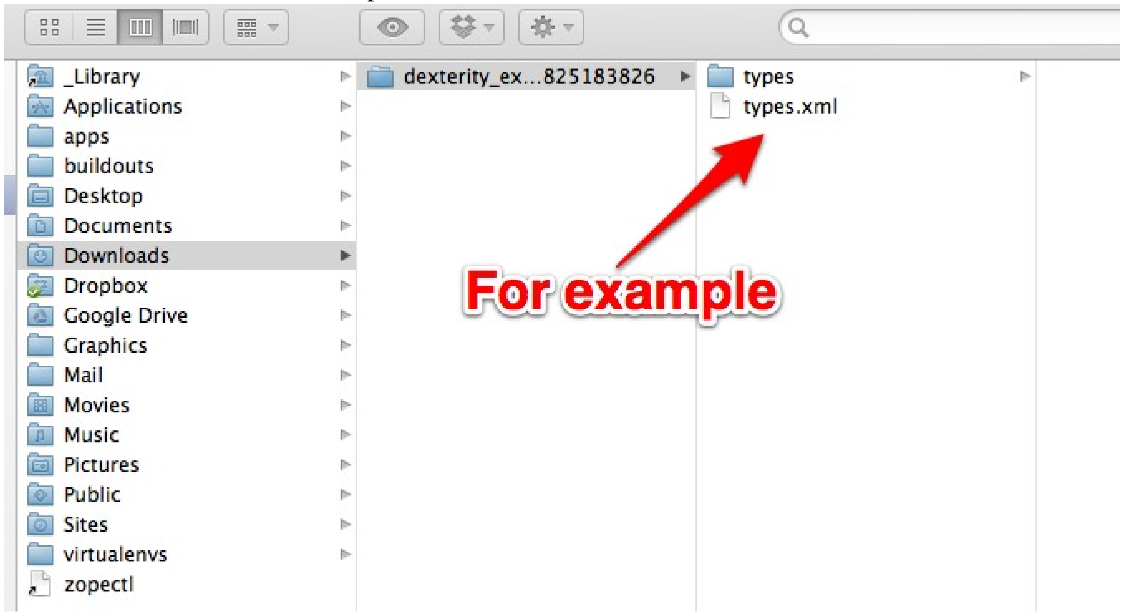
Navigate back to the dexterity content type panel or go directly to `http://localhost:8080/Plone/@@dexterity-types`

Check the TODO item and then click on export type profile to download the type. If you don't check anything, it won't do anything and there is currently no error message so don't be surprised.



select	Type Name	Description	# of items
☐	TODO Item	Something ne	4

This will start a download to your machine. Navigate to the download directory and unzip the contents of the file that was downloaded. Here is an example of what it will look like:



We need to take types.xml and the types folder, and save it in our base product. You can use your finder or explorer to drag and drop, or use the command line. I'll use command line as an example but feel free to improvise. You want to move the files into your default product profile. What's a profile? Don't worry about it. Sit back, relax, and finish the tutorial. You will move the files into

```
tutorial.todoapp/src/tutorial/todoapp/profiles/default
```

Warning: There is a bug in Plone 4.3 that makes the import process brake when parsing XMLs that the export tool exports. To make it work we need to remove the *comment* line from `types.xml`:

```
<!---*- extra stuff goes here -*--->
```

Anytime you perform some sort of configuration export from Plone to a custom product, you will put the XML files in the profiles/default folder. Every time you make changes to your types, you should re-export and save into the same location. Now, when the next person installs the add-on, they wil have the type already there!

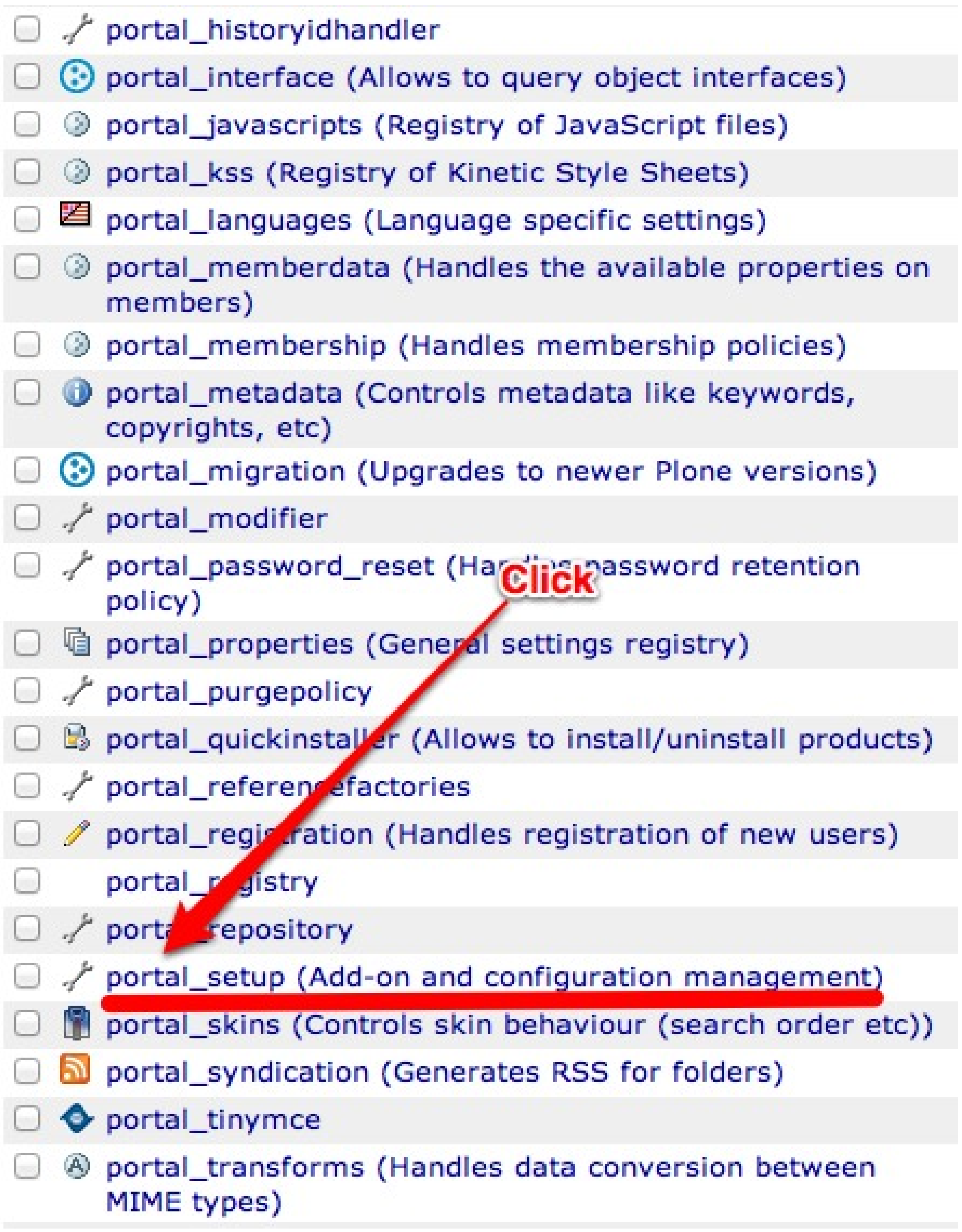
Dependencies

Before we continue we need to tell Plone that whenever we install *tutorial.todoapp* we want it to also pull in the Dexterity package, so our content type is working happily. We do that by adding the following lines to `profiles/default/metadata.xml`, inside the `<metadata>` tag.

```
<dependencies>
  <dependency>profile-plone.app.dexterity:default</dependency>
</dependencies>
```

Exporting todo_item_workflow

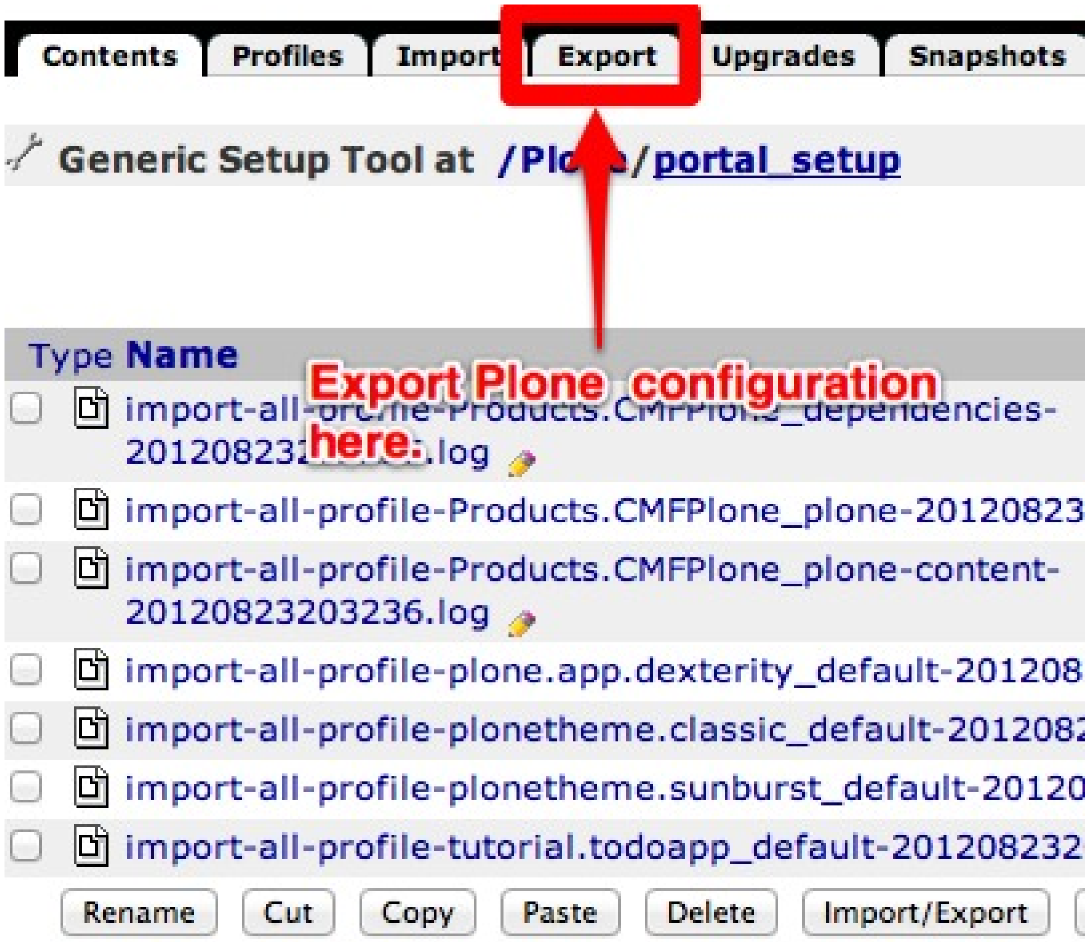
Exporting a workflow is very similar to exporting a Dexterity type. It just takes a little bit more navigating and a trip to the ZMI. To export the workflow, navigate to the root of the ZMI by gong to `http://localhost:8080/Plone/manage_main`. From there, head into the *portal_setup* tool:



portal_historyidhandler
portal_interface (Allows to query object interfaces)
portal_javascripts (Registry of JavaScript files)
portal_kss (Registry of Kinetic Style Sheets)
portal_languages (Language specific settings)
portal_memberdata (Handles the available properties on members)
portal_membership (Handles membership policies)
portal_metadata (Controls metadata like keywords, copyrights, etc)
portal_migration (Upgrades to newer Plone versions)
portal_modifier
portal_password_reset (Handles password retention policy)
portal_properties (General settings registry)
portal_purgepolicy
portal_quickinstaller (Allows to install/uninstall products)
portal_referencefactories
portal_registration (Handles registration of new users)
portal_registry
portal_repository
portal_setup (Add-on and configuration management)
portal_skins (Controls skin behaviour (search order etc))
portal_syndication (Generates RSS for folders)
portal_tinymce
portal_transforms (Handles data conversion between MIME types)

WARNING: The following User Interface is not recommended for children under 18.

In the *portal_setup* tool, click on the export tab.

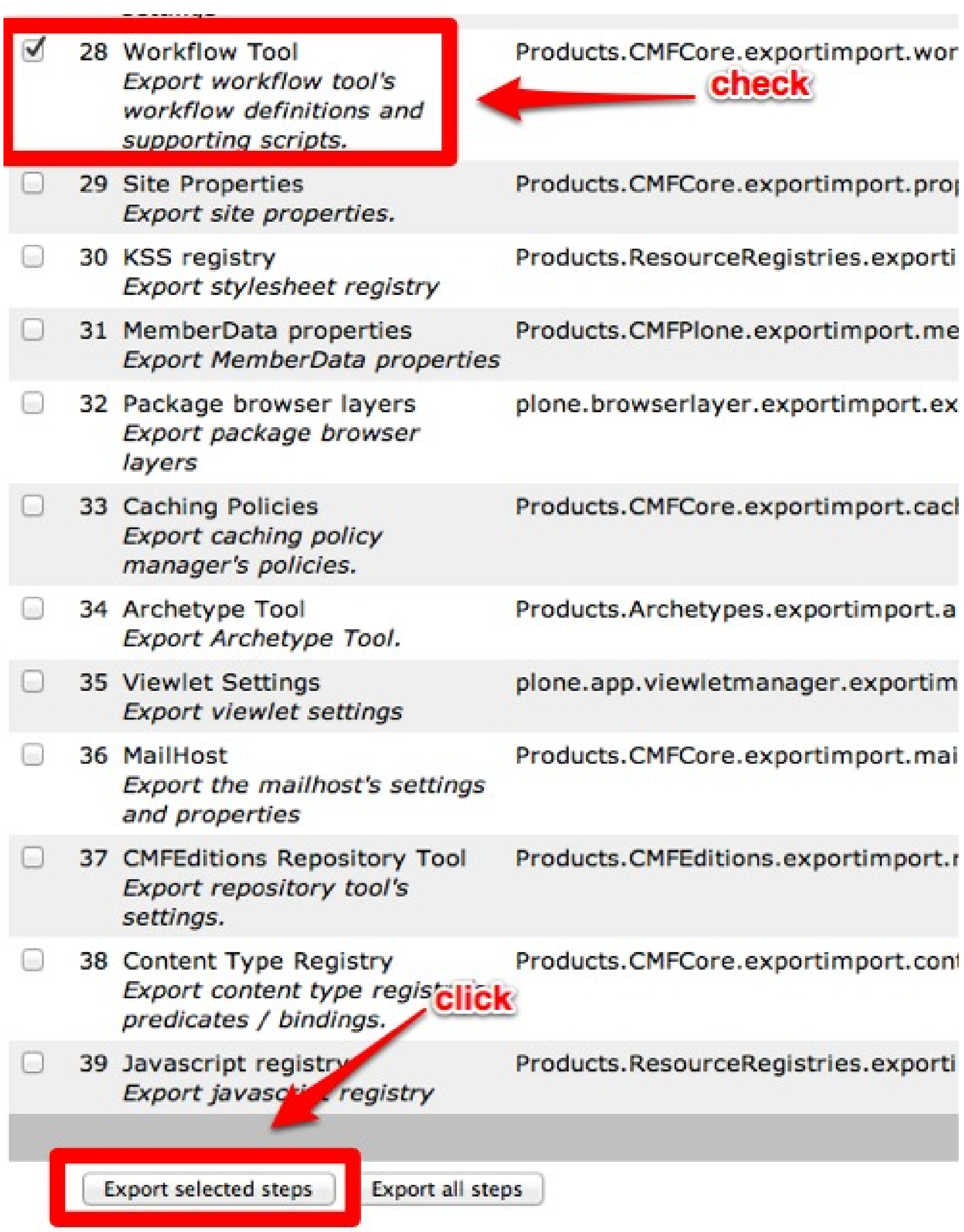


Generic Setup Tool at /Plone/portal_setup

Type Name

import-all-profile-Products.CMFPlone_dependencies-20120823...log
import-all-profile-Products.CMFPlone_plone-20120823
import-all-profile-Products.CMFPlone_plone-content-20120823203236.log
import-all-profile-plone.app.dexterity_default-201208
import-all-profile-plonetheme.classic_default-201208
import-all-profile-plonetheme.sunburst_default-20120
import-all-profile-tutorial.todoapp_default-201208232

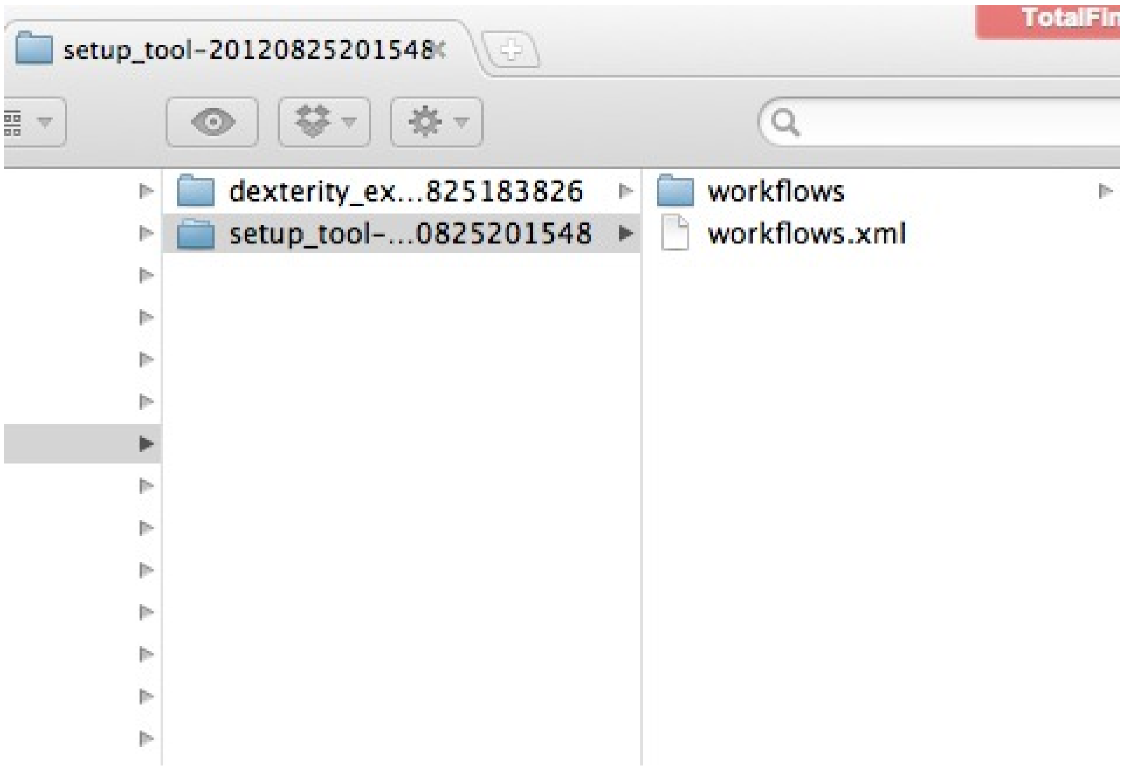
There are a LOT of things that you can export here, but that is for a different tutorial. For now, find export item #27 called `Workflow Tool`, check the box to the left. Then scroll all the way to the bottom and `Export selected steps`.



✓	28	Workflow Tool *Export workflow tool's workflow definitions and supporting scripts.*	Products.CMFCore.exportimport.wor
☐	29	Site Properties *Export site properties.*	Products.CMFCore.exportimport.pro
☐	30	KSS registry *Export stylesheet registry*	Products.ResourceRegistries.exporti
☐	31	MemberData properties *Export MemberData properties*	Products.CMFPlone.exportimport.me
☐	32	Package browser layers *Export package browser layers*	plone.browserlayer.exportimport.ex
☐	33	Caching Policies *Export caching policy manager's policies.*	Products.CMFCore.exportimport.cac
☐	34	Archetype Tool *Export Archetype Tool.*	Products.Archetypes.exportimport.a
☐	35	Viewlet Settings *Export viewlet settings*	plone.app.viewletmanager.exportim
☐	36	MailHost *Export the mailhost's settings and properties*	Products.CMFCore.exportimport.mai
☐	37	CMFEditions Repository Tool *Export repository tool's settings.*	Products.CMFEditions.exportimport.r
☐	38	Content Type Registry *Export content type regis predicates / bindings.*	Products.CMFCore.exportimport.con
☐	39	Javascript registry *Export javasc registry*	Products.ResourceRegistries.exporti

Just like the Dexterity content type, you will want to untar the downloaded folder, and move into your default profile folder.

In that download you should have a file called `workflows.xml` and a folder called `workflows` like below. You will move both of them to the default profile.



Place all of these files in your profile at

```
tutorial.todoapp/src/tutorial/todoapp/profiles/default
```

Now, this export exported the entire configuration for all workflows in your site. But you are only interested in the `todo_item_workflow` configuration and you don't want to change configuration for other workflows. So, first, remove all other workflow definitions (XML files).

```
rm -rf tutorial.todoapp/src/tutorial/todoapp/profiles/default/workflows/comment_review_workflow
rm -rf tutorial.todoapp/src/tutorial/todoapp/profiles/default/workflows/folder_workflow
rm -rf tutorial.todoapp/src/tutorial/todoapp/profiles/default/workflows/intranet_folder_workflow
rm -rf tutorial.todoapp/src/tutorial/todoapp/profiles/default/workflows/intranet_workflow
rm -rf tutorial.todoapp/src/tutorial/todoapp/profiles/default/workflows/one_state_workflow
rm -rf tutorial.todoapp/src/tutorial/todoapp/profiles/default/workflows/plone_workflow
rm -rf tutorial.todoapp/src/tutorial/todoapp/profiles/default/workflows/simple_publication_workf
```

Secondly, remove all non-todoitem-related stuff from `workflows.xml`. In the end the file should look like this:

```xml
<?xml version="1.0"?>
<object name="portal_workflow" meta_type="Plone Workflow Tool">
 <object name="todo_item_workflow" meta_type="Workflow"/>
 <bindings>
  <type type_id="todo_item">
   <bound-workflow workflow_id="todo_item_workflow"/>
  </type>
 </bindings>
</object>
```

And you are done! Congratulations on the birth of your new product!

1.6.3 Tests

Alright, tests! Considered a pain and a nuisance by some but loved by all who do it. If you want your code to be solid and your site to be stable, tests are a great way to get there.

The package you have on your filesystem is already configured to give you a test-runner so you can immediately go and run it – obviously you have no tests, but at least you try if your test runner works.

```
tutorial.todoapp$ make tests
Total: 0 tests, 0 failures, 0 errors in 0.000 seconds.
```

Note: you do *NOT* need to stop your Plone instance in order to run tests. They will peacefully co-exist.

Good, the next thing to do is to add tests. Go to tutorial.todoapp repo on GitHub and copy/paste (or download) all files from the `src/tutorial/todoapp/tests` folder to your local `src/tutorial/todoapp/tests` folder. You can also get the tests with git:

```
$ git branch --track chapter2 origin/chapter2  # tell git what chapter2 is
$ git checkout chapter2 src/tutorial/todoapp/tests  # get tests
```

This folder will contain your test files:

- **test_setup.py**

  This module contains tests that check if your package was successfully installed and configured. Tests in here are concerned with XML files you have in the `profiles/default` folder.

- **test_todo_item.py**

  And finally a module that contains tests for your custom content-type.

We will not go into details of what each test does as we believe the test code and its comments are in themselves informative and we will rather encourage you to go through all tests, try to understand what they do, maybe change something and see what happens, etc.

Remember that you run tests with `make tests` and you should get an output that looks somewhat like this:

```
tutorial.todoapp$ make tests
[...snip...]
Set up tutorial.todoapp.tests.base.TodoAppLayer:Integration in 0.000 seconds.
Running:

Ran 11 tests with 0 failures and 0 errors in 9.782 seconds.
Tearing down left over layers:
Tear down tutorial.todoapp.tests.base.TodoAppLayer:Integration in 0.000 seconds.
Tear down tutorial.todoapp.tests.base.TodoAppLayer in 0.004 seconds.
Tear down plone.app.testing.layers.PloneFixture in 0.164 seconds.
Tear down plone.testing.z2.Startup in 0.012 seconds.
Tear down plone.testing.zca.LayerCleanup in 0.004 seconds.
```

Also, remember that whenever you run `make` your tests are gonna be run too.

1.6.4 Troubleshooting

If something goes wrong you can always go to GitHub and see how the code for chapter 2 should look like and compare this to what you have on your local machine.

WARNING: If you are reading this on GitHub, DON'T! Read it on ReadTheDocs:
http://tutorialtodoapp.readthedocs.org/en/latest/chapter_3.html so you

have working references and proper formatting.

1.7 Chapter 3: Custom View

In this chapter you will learn how to add a custom view – in our case a listing of Todo Items.

1.7.1 View class

Let's start by adding the view class. You can go to tutorial.todoapp repo on GitHub and copy over code from `src/tutorial/todoapp/todo.py` to your local computer or just use git:

```
$ git branch --track chapter3 origin/chapter3  # tell git what chapter 3 is
$ git checkout chapter3 src/tutorial/todoapp/todo.py
```

We also need to tell Plone to display this view in the *display* drop-down menu for Folders so we will later be able to set our view as a default display view for our Todo folder. Let's do that by using git to get a version of `Folder.xml` and put it in `src/tutorial/todoapp/profiles/default/types`.

```
$ git checkout chapter3 src/tutorial/todoapp/profiles/default/types/Folder.xml
```

1.7.2 View template

Now that we have a class we can also add the template. Go to tutorial.todoapp repo on GitHub and copy over code from `src/tutorial/todoapp/templates/todo.pt` to your local computer or, again, use git.

```
$ git checkout chapter3 src/tutorial/todoapp/templates/todo.pt
```

The template uses the ZPT syntax, read more about it here.

1.7.3 Static resources

The template displays different icons for different workflow states of your Todo Items. We need to add these icons to your package:

1. Download `open.png` and `completed.png` from GitHub (they are in `src/tutorial/todoapp/static`) into a new folder on your local computer: `src/tutorial/todoapp/static`. You can use git again if you don't like manual work.

   ```
   $ git checkout chapter3 src/tutorial/todoapp/static
   ```

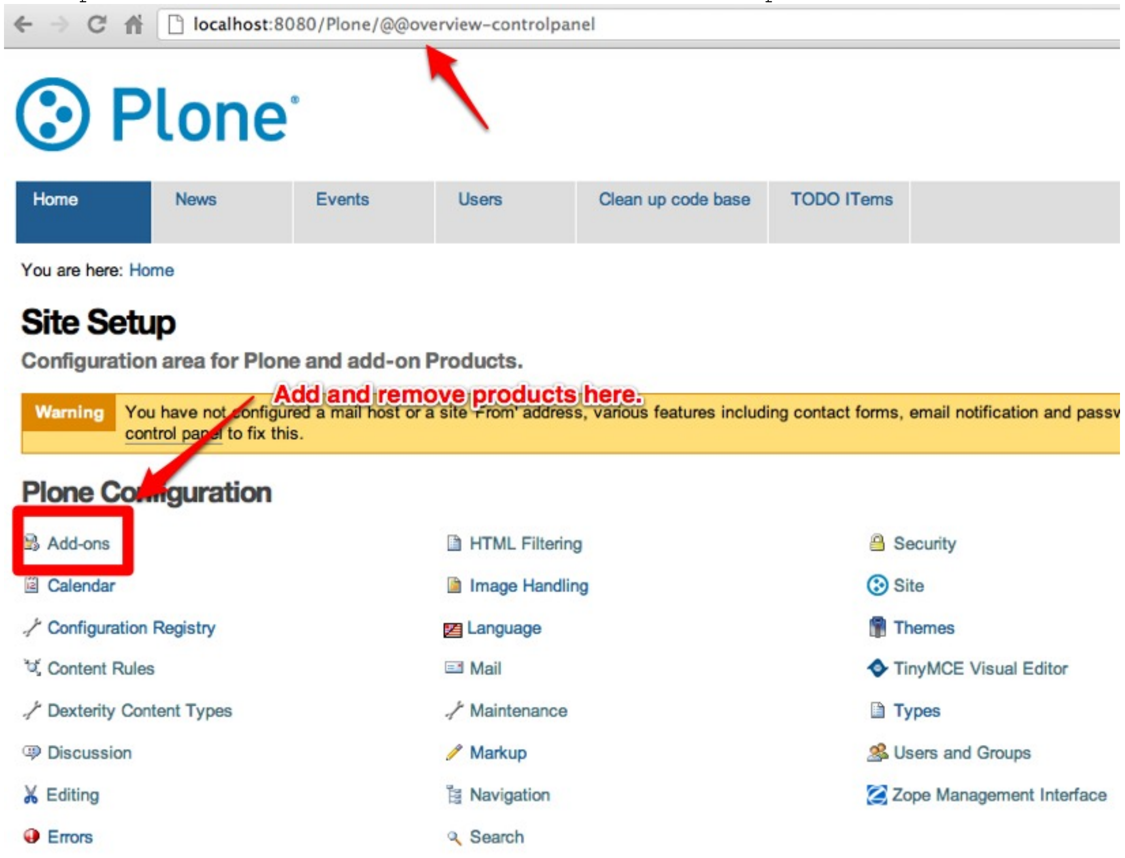
2. Tell Zope that this `static` folder contains static resources (icons, CCS files, JavaScript files, etc.) by adding the following lines to `src/tutorial/todoapp/configure.zcml` inside the `<configure` tag:

   ```
   <!-- Publish static files -->
   <browser:resourceDirectory
       name="tutorial.todoapp"
       directory="static" />
   ```

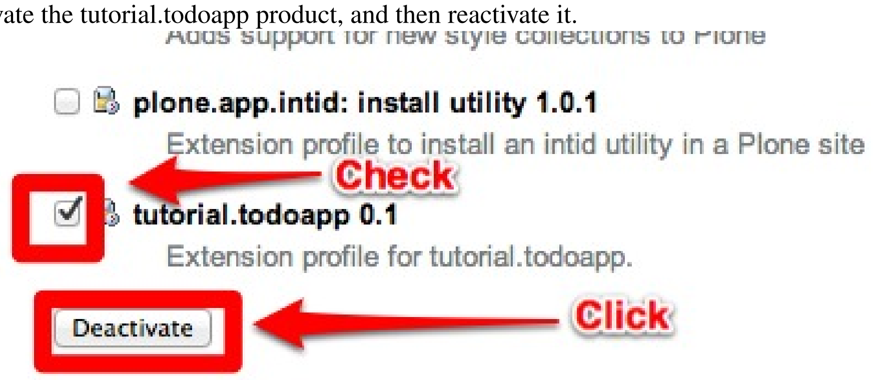
After restarting your Zope server, files in your `static` folder will be available on a standard URL: `http://localhost:8080/Plone/++resource++tutorial.todoapp/<filename>`

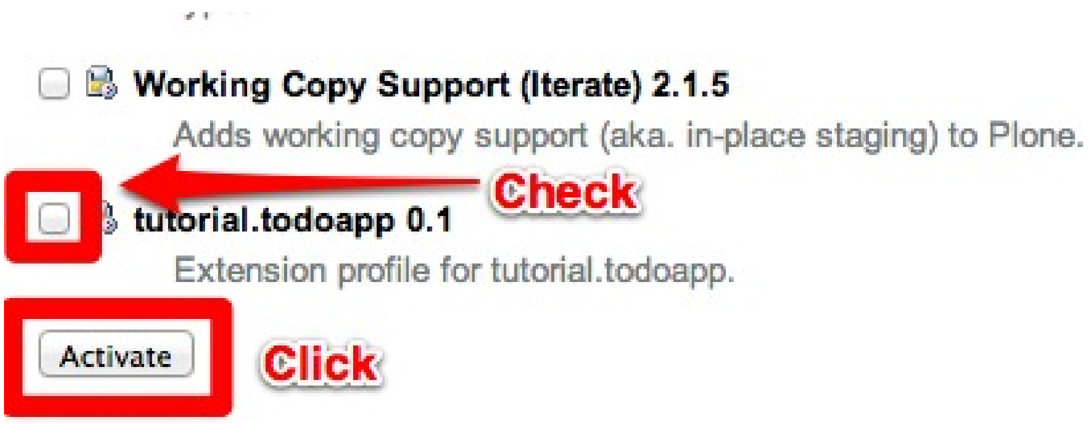
1.7.4 Try it out

Because the XML configuration of our product has change, we need to reinstall the product. This is accomplished by `deactivating` and `reactivating` the product. Navigate to the add-ons manager or go directly to `http://localhost:8080/Plone/@@overview-controlpanel`.



Deactivate the tutorial.todoapp product, and then reactivate it.


Adds support for new style collections to Plone
plone.app.intid: install utility 1.0.1
Extension profile to install an intid utility in a Plone site
Check
tutorial.todoapp 0.1
Extension profile for tutorial.todoapp.
Deactivate
Click



Working Copy Support (Iterate) 2.1.5

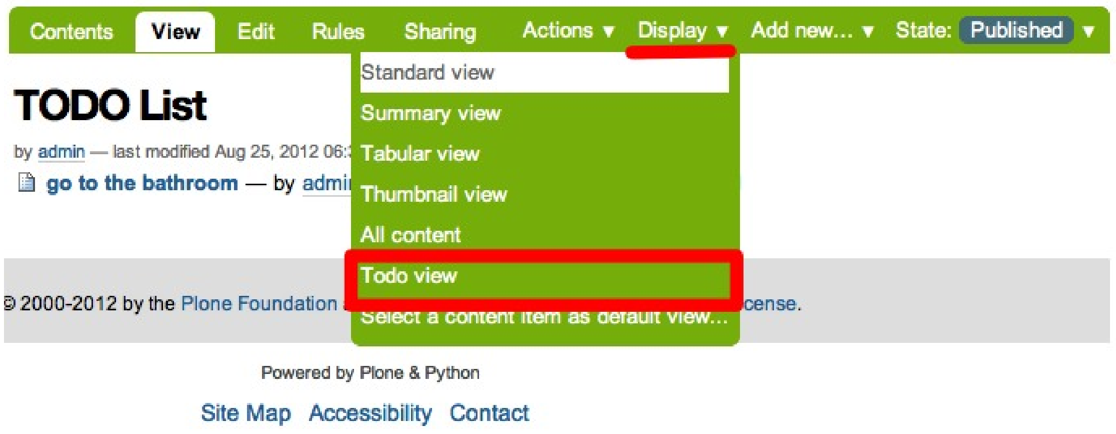
Adds working copy support (aka. in-place staging) to Plone.

tutorial.todoapp 0.1

Extension profile for tutorial.todoapp.

Note that every time you make a change to the xml files, by exporting or manual edit, you must reactivate the product for the changes to take effect!

Now, we apply the new view to the folder holding our todo items. Navigate to the folder you created in chapter 1, and update the display.



TODO List

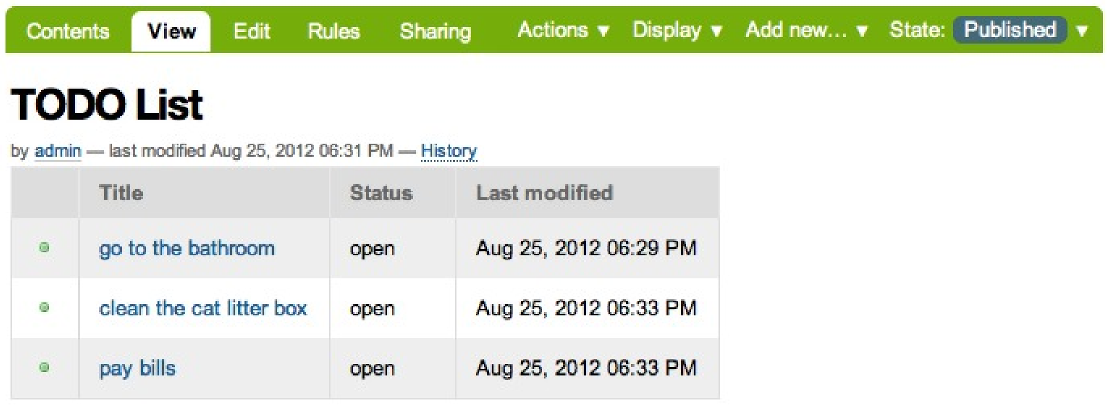
by admin — last modified Aug 25, 2012 06:

go to the bathroom — by admi

© 2000-2012 by the Plone Foundation cense.

Powered by Plone & Python

Celebrate!



TODO List

by admin — last modified Aug 25, 2012 06:31 PM — History

	Title	Status	Last modified
●	go to the bathroom	open	Aug 25, 2012 06:29 PM
●	clean the cat litter box	open	Aug 25, 2012 06:33 PM
●	pay bills	open	Aug 25, 2012 06:33 PM

If the de-activate / activate does not work you may need to restart Plone instance to see the changes.

1.7.5 Tests

Cool, so you have verified that your code works through the browser and it's time to add tests to make sure your code keeps on working in the future.

First add the following snippet to `test_setup.py` to verify that your Folders have the `todo` view on the *Display* drop-down menu.

```
# types/Folder.xml
def test_folder_available_layouts(self):
    """Test that our custom display layout (@@todo) is available on folder.

    Also make sure that layouts that come with Plone out-of-the-box are
    also still there.
    """
    layouts = self.portal.folder.getAvailableLayouts()
    layout_ids = [id for id, title in layouts]

    # out-of-the-box layouts are still there
    self.assertIn('folder_listing', layout_ids)
    self.assertIn('folder_summary_view', layout_ids)
    self.assertIn('folder_tabular_view', layout_ids)
    self.assertIn('atct_album_view', layout_ids)
    self.assertIn('folder_full_view', layout_ids)

    # our custom one
    self.assertIn('todo', layout_ids)
```

If you haven't already downloaded it, add a new test module: `test_todo_view.py`. Download it from GitHub, put and it in your `tests` folder and run tests. Feel free to fiddle around with it to see what it does. As always, you can use git to get the file.

```
$ git checkout chapter3 src/tutorial/todoapp/tests/test_todo_view.py
```

1.7.6 Troubleshooting

If something goes wrong you can always go to GitHub and see how the code for chapter 3 should look like and compare this to what you have on your local machine.

WARNING: If you are reading this on GitHub, DON'T! Read it on ReadTheDocs:
http://tutorialtodoapp.readthedocs.org/en/latest/chapter_3.html so you
have working references and proper formatting.

1.8 Chapter 4: Bling-bling

As a reward for making it all the way to the end, we will help you add some fancy features to your project, otherwise known as bling and that means having to write JavaScript. Fortunately Plone comes with jQuery so we can easily integrate.

The final part of this tutorial will allow users to check and un-check items on their todo list without having to load a new page request. Note that by developing the functionality in this order, 100% of the functionality of the application remains working even when javascript is disabled. Win!

1.8.1 AJAX view

Before we add front-end bling, we need some code that can handle these requests coming in. Let's create a simple view that will update the object in context to a new state. Go to GitHub and copy the code for `WorkflowTransition` class in `todo.py`. This class represents a view that our AJAX code will call. You can also get the code with git, however note that now we are checking out code from master, as Chapter 4 is the last chapter and its code is in the master branch.

```
$ git checkout master src/tutorial/todoapp/todo.py
```

Take a look at the `WorkflowTransition` class and comments around the code. There are a couple of things to point out specific to this setup:

```
grok.context(Container)
```

Tells us that this view should be called in the context of a Dexterity Container item. So if you try to go to this view from the portal root or anywhere in the site that is not a Dexterity item, Plone will return a 404 - not found error. By default all Dexterity types that you create TTW are based on the Dexterity Container base class.

```
grok.name('update_workflow')
```

This tells us on which URL the view will be available on. In this case, on `<url_to_plone_content_object>/update_workflow`.

```
def render(self):
```

`render` is a special function that must be used. It is where all of the code must go when used with grok directives. This is the main block of code that will be executed.

```
transition = self.request.form.get('transition', '')
```

`self.request` is set by the base class, and anything based on BrowserView will have access to this variable. All of GET/POST parameters will be stored in `self.request.form`.

```
self.request.response.setHeader(
    'Content-Type', 'application/json; charset=utf-8')
return json.dumps(results)
```

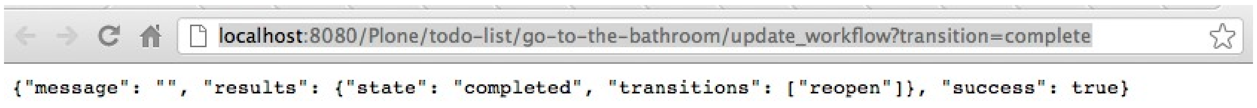
When working with JSON, it's not *required* to set the header content type, but when used with certain jQuery calls it is expected to have the header set correctly. If you don't set this, it will sometimes work and sometimes not. Get used to setting it!

Additionally, we return the result serialized as json by default. For making and testing JSON web service calls, keep in mind that they should do exactly one thing and no more. This makes it easy to integrate with Javascript and VERY easy to test. We'll see later on how easy it is to test this view.

Furthermore, before taking the plunge to wire up JavaScript, go directly to the url and test the change. For example, if you have an item at `http://localhost:8080/Plone/todo-list/go-to-the-bathroom`, you can test the view by appending the view name and GET variables to the end of the item's url. However, you first need to restart your Zope first, so your Python files get reloaded!

```
http://localhost:8080/Plone/todo-list/go-to-the-bathroom  + update_workflow?transition = complete
```

```
http://localhost:8080/Plone/todo-list/go-to-the-bathroom/update_workflow?transition=complete
```


localhost:8080/Plone/todo-list/go-to-the-bathroom/update_workflow?transition=complete

```
{"message": "", "results": {"state": "completed", "transitions": ["reopen"]}, "success": true}
```

For extra clarity: if you are not an expert in python, plone, AND javascript, I highly recommend integrating bling bling in the following order:

1. Write base view and **passing** test cases
2. Test views in browser
3. Make ajax interactive

Starting with bling from the start will only bring you pain.

1.8.2 Custom JavaScript

Now that we know the `update_workflow` view is working, let's add some AJAX handling on the top of it. Checkout the Javascript file and a JavaScript registry file into your working directory:

```
git checkout master src/tutorial/todoapp/static/todoapp.js
git checkout master src/tutorial/todoapp/profiles/default/jsregistry.xml
```

`jsregistry.xml` contains all configuration needed to tell Plone how it should register and use our JavaScript. It has a lot of options that are pretty self explanatory (if you think like a machine).

```
<?xml version="1.0"?>
<object name="portal_javascripts">
  <javascript
    cacheable="True"
    compression="safe"
    conditionalcomment=""
    cookable="True"
    enabled="True"
    expression=""
    inline="False"
    id="++resource++tutorial.todoapp/todoapp.js" />
</object>
```

1.8.3 Trying it out!

Holy moley you made it! Restart Zope (to reload Python files), reactivate the product (to reimport XML files), do a hard reload in your web browser (to clear any caches) and check out your todo list. The todo items should toggle between complete and incomplete without the page reloading. Sweet!

1.8.4 Tests

As always, let's add tests! First add the following snippet to `test_setup` to verify that your JavaScript is registered in Plone.

```
# jsregistry.xml
def test_js_registered(self):
    """Test that todoapp.js file is registered in portal_javascript."""
    resources = self.portal.portal_javascripts.getResources()
    ids = [r.getId() for r in resources]
    self.assertIn('++resource++tutorial.todoapp/todoapp.js', ids)
```

Lastly, add a new test module: `test_workflow.py`. Download it from GitHub, put and it in your `tests` folder and run tests. Then fiddle around with it to see what it does. As always, you can use git to get the file.

```
$ git checkout master src/tutorial/todoapp/tests/test_workflow.py
```

1.8.5 The end

This concludes the Todo app in Plone tutorial. Congratulations! Now it's time to checkout other tutorials and documentation available on developer.plone.org!

1.8.6 Troubleshooting

If something goes wrong you can always go to GitHub and see how the code in master should look like and compare this to what you have on your local machine.

CHAPTER 2

Developer Documentation

Information on how to contribute to this tutorial. Note that all code should follow plone.api code conventions.

2.1 Releasing a new version

Releasing a new version of *tutorial.todoapp* involves the following steps:

1. Create a git tag for the release.
2. Push the git tag upstream to GitHub.
3. Generate a distribution file for the package.
4. Upload the generated package to Python Package Index (PyPI).

2.1.1 Checklist

Before every release make sure that:

1. You have documented your changes in the `HISTORY.rst` file.
2. You have modified the version identifier in `setup.py` to reflect the new release.
3. You have confirmed that the package description (generated from `README.rst` and others) renders correctly by running `bin/longtest`.
4. You have committed all changes to the git repository and pushed them upstream.
5. You have the working directory checked out at the revision you wish to release.

2.1.2 Actions

For help with releasing we use `jarn.mkreleaser`. It's listed as a dependency in `setup.py` and should already be installed in your local bin:

```
$ bin/mkrelease -d pypi -pq ./
```

Note: In order to push packages to PyPI you need to have the appropriate access rights to the package on PyPI and you need to configure your PyPI credentials in the `~/.pypirc` file, e.g.:

```
[distutils]
index-servers =
  pypi

[pypi]
username = fred
password = secret
```

2.1.3 Example

In the following example we are releasing version 0.1 of *tutorial.todoapp*. The package has been prepared so that `setup.py` contains the version `0.1`, this change has been committed to git and all changes have been pushed upstream to GitHub:

```
# Check that package description is rendered correctly
$ bin/longtest

# Make a release and upload it to PyPI
$ bin/mkrelease -d pypi -pq ./
Releasing tutorial.todoapp 0.1
Tagging tutorial.todoapp 0.1
To git@github.com:collective/tutorial.todoapp.git
* [new tag]         0.1 -> 0.1
running egg_info
running sdist
warning: sdist: standard file not found: should have one of README, README.txt
running register
Server response (200): OK
running upload
warning: sdist: standard file not found: should have one of README, README.txt
Server response (200): OK
done
```

Note: Please ignore the sdist warning about README file above. PyPI does not depend on it and it's just a bug in setupools (reported and waiting to be fixed).

2.2 Changelog

2.2.1 1.1 (2013-07-04)

- Instructions for preparing the environment on various OSes. [ielectric, zupo]
- Proof-reading the tutorial text. [ielectric, zupo]
- Use latest best practices from bobtemplates.niteoweb. [zupo]
- Use Plone 4.3. [zupo]

2.2.2 1.0 (2012-09-11)

- Acted as guinea pigs and went through the entire tutorial slowly and thoroughly. [matejc, plamut]

- AJAXifying the @@todo view. [eleddy]
- The @@todo BrowserView for listing Todo Items. [zupo]
- Tests for GenericSetup exports of content-type and workflow. [zupo]
- TTW part of the tutorial, loads of screenshots. [eleddy]
- Skeleton. [zupo]

2.3 License (3-clause BSD)

CHAPTER 3

Indices and tables

- *genindex*
- *modindex*
- *search*