Asko Soukka, asko.soukka@iki.fi
github.com/datakurre

Instant features with advanced Plone themes

JYVÄSKYLÄN YLIOPISTO
UNIVERSITY OF JYVÄSKYLÄ

What if. . .

Case of the day: Wall of images

- Resource bundles
- Folderish content type
- Custom view template for Wall of images with Masonry.js layout
- Workflow for accepting anonymous submissions
- Image content type for anonymous submissions
- Add permission for restricting anonymous visitor submission
- Content rule for thanking about submission
- Review portlet for listing the pending submissions
- i18n with l10n message catalog

Python packages vs. theme?

Python packages

- One package for new JS
- Another for new types
- Customer specific theming of components in theme package
- Configuring everything in policy package
- Restarting Plone

Advanced theme

- Everything in a single zipped Plone theme package
- Upload through Plone control panel or using npm package `plonetheme-upload`

Plone customization features

- Configurable registry
- Structured content types
- Content type behaviors
- State based workflows
- Roles and permissions
- All types of portlets
- Content rules on events
- Restricted templates
- Restricted Python scripts
- Static frontend resources
- Diazo transform rules
- . . .

Plone customization features

- Configurable registry
- Structured content types
- Content type behaviors
- State based workflows
- Roles and permissions
- All types of portlets
- Content rules on events
- **Restricted templates**
- **Restricted Python scripts**
- Static frontend resources
- Diazo transform rules
- . . .

Restricted Python?

”Restricted Python provides a safety net for programmers who don’t know the details of the Zope/Plone security model.

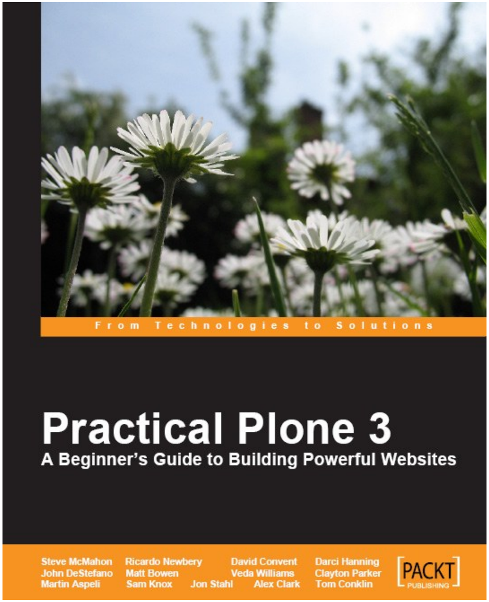
It is important that you understand that the safety net is not perfect: it is not adequate to protect your site from coding by an untrusted user.”

– Steve McMahon, collectice.ambidexterity README

Issues with Restricted Python

Not only ponies and unicorns...

- **API is** `restrictedTraversed`**, not imported**
- **API is not always complete**
- **API is not always up-to-date**
- **API is scattered around the ecosystem**
- `plone.api` **is not currently designed or available by default to be called from Restricted Python**



From Technologies to Solutions

Practical Plone 3

A Beginner's Guide to Building Powerful Websites

Steve McMahon
John DeStefano
Martin Aspeli
Ricardo Newbery
Matt Bowen
Sam Knox
Jon Stahl
David Convent
Veda Williams
Alex Clark
Darci Hanning
Clayton Parker
Tom Conklin

PACKT PUBLISHING

Products.DocFinderTab

”through-the-web”

$=$

technical debt

Advanced Plone themes

- Theme with any supported customizations
- Editable using Plone theme editor
- Rollbacks with Zope2 undo form
- Zip-exportable and importable from theme editor
- Supports ”TTW” and ”file system" development
- Exports can be version controlled
- Exports can be acceptance tested

collective.themesitesetup

Theme activation or update

- Imports GS profile steps
- Updates DX models
- Registers permissions
- Registers l10m messages
- Copies resources into `portal_resources`

Theme deactivation

- Imports GS profile steps
- Unregisters added custom permissions
- Unregisters added custom l10n messages

After ”TTW” development

- `@@export-site-setup`

collective.themefragments

View templates: `./fragments/foobar.pt`

- **Can be injected as fragments using Diazo rules**
- **Can be configured as a default view for any content type**
- **Can be used as a local view by setting** `layout` **attribute**

Python scripts: `./fragments/foobar.py`

- **Can provide view methods for view templates with matching base name (**`tal:define="data view/getData"`**)**

Faster iterations

=

More iterations

Let’s get our hands dirty. . .

Creating Wall of images

- Initial configuration was made through Plone site setup
- Configuration was exported into a new theme using the export view `++theme++.../@@export-site-setup`
- Theme was zip-exported from theming control panel
- Content type schemas were exported from Dexterity editor
- Development was completed on a regular file system buildout using file system resources directory

Structure of Wall of images

```
./bundles/
./fragments/
./install/types/
./install/workflows/
./install/
./locales/LC_MESSAGES/*/
./models/
./index.html
./manifest.cfg
./preview.png
./rules.xml
./scripts.js
./styles.css
```

Custom frontend bundles 1/2

```
./bundles/imagesloaded.pkgd.min.css
./bundles/imagesloaded.pkgd.min.js
./bundles/masonry.pkgd.min.css
./bundles/masonry.pkgd.min.js
```

```
(function() { var require, define;
// ...
// AMD packaged JS distributions must be wrapped
// so that Plone require.js define is undefined
// during their load
// ...
})();
```

Custom frontend bundles 2/2

`./install/registry.xml`

```
<records prefix="plone.bundles/imagesloaded-js"
         interface="...interfaces.IBundleRegistry">
  <value key="depends">plone</value>
  <value key="jscompilation">++theme++...js</value>
  <value key="csscompilation">++theme++...css</value>
  <value key="last_compilation">2017-10-06 00:00:00
  </value>
  <value key="compile">False</value>
  <value key="enabled">True</value>
</records>
```

Custom content types

```
./install/types/wall_of_images.xml
./install/types/wall_of_images_image.xml
./install/types.xml
```

```
<object name="portal_types">
 <object name="wall_of_images"
         meta_type="Dexterity FTI"/>
 <object name="wall_of_images_image"
         meta_type="Dexterity FTI"/>
</object>
```

```
./models/wall_of_images.xml
./models/wall_of_images_image.xml
```

Custom content types

```
<model xmlns:...="..." i18n:domain="plone">
  <schema>
    <field
        name="title" type="zope.schema.TextLine">
      <title i18n:translate="">Title</title>
    </field>
    <field
        name="image"
        type="plone.namedfile.field.NamedBlobImage">
      <title i18n:translate="">Image</title>
    </field>
  </schema>
</model>
```

Custom views 1/3

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:...="..."
      lang="en"
      metal:use-macro=".../macros/master"
      i18n:domain="plone">
<body>
<metal:main fill-slot="main">
  <metal:content-core define-macro="content-core">
  </metal:content-core>
</metal:main>
</body>
</html>
```

Custom views 2/3

./fragments/wall-of-images.pt

```
<div class="wall-of-images container-fluid"
     tal:define="items context/@@contentlisting">
  <tal:image tal:repeat="item items">
    <img tal:define="obj item/getObject;
        scale_func obj/@@images;
        scaled_image python:scale_func.scale('image',
 ↪  scale='preview')"
      tal:replace="structure
       ↪  python:scaled_image.tag()"
      tal:on-error="string:error" />
  </tal:image>
</div>
```

Custom views

```
./install/types/wall_of_images.pt

...
<property name="default_view">
    ++themefragment++wall-of-images</property>
<property name="view_methods">
  <element value="++themefragment++wall-of-images"/>
</property>
...
```

Any themefragment can be configured as content object view by manually setting `layout` property of the content object.

Custom l10n messages

```
./locales/fi/LC_MESSAGES/plone.po
./locales/en/LC_MESSAGES/plone.po
```

```
msgid "Close from submissions"
msgstr "Sulje osallistuminen"

msgid "Open for submissions"
msgstr "Avaa osallistumiselle"
```

When theme is developed on file system, normal i18n tools can be used for messages extraction and catalog synchronization (e.g. i18ndude).

Custom permissions

./manifest.cfg

```
[theme:genericsetup]
permissions =
    demotheme.addImage  Wall of Images: Add Image
```

./install/types/wall_of_images_image.xml

```
<object name="wall_of_images_image" ...="..">
  ...
  <property name="add_permission">
      demotheme.addImage</property>
  ...
</object>
```

Custom workflows

```
./install/workflows.xml
./install/simple_publication_with_submission_workflow/
./install/wall_of_images_workflow/
```

```
<type type_id="wall_of_images">
  <bound-workflow workflow_id="..."/>
</type>
```

```
<dc-workflow workflow_id="...">
  ...
  <permission>Wall of Images: Add Image</permission>
  ...
</dc-workflow>
```

Custom content rules

./install/contentrules.xml

```
<contentrules>
  <rule name="rule-image-thank-you"
        title="Thank visitor from submission"
        cascading="False"
        description="Thanks visitor after submission"
        event="...IObjectAddedEvent"
        stop-after="False"
        enabled="True">...</rule>
  <assignment
      name="rule-image-thank-you" bubbles="True"
      enabled="True" location=""/>
</contentrules>
```

Final touch with Diazo-bundles

./manifest.cfg

```
production-css = /++theme++demotheme/styles.css
production-js = /++theme++demotheme/scripts.js
```

./scripts.js

```
jQuery(function($) {
  $('.wall-of-images').imagesLoaded(function() {
    $('.wall-of-images').masonry({
      itemSelector: 'img',
      percentPosition: true
    });
  });
});
```

Questions?

github.com/datakurre/ploneconf2017

What about Mosaic?

- Theme site setup can populate Mosaic site and content layout resource directories from theme
- Theme fragment tile for Plone Mosaic can render selected fragment with
  - fragment-specific readable title
  - fragment-specific XML schema based configuration form
  - fragment-specific view permission
  - fragment-specific caching rule

What about Webpack?

- Bundles in theme can be built with Webpack
- Diazo bundle in theme can be built with Webpack
- All frontend resources can be built with Webpack
  - `plonetheme.webpacktemplate`
  - `plonetheme-webpack-plugin`
  - `plonetheme-upload`

Bonus: Custom JS widgets

./models/my_content_type.xml

```
...
<field name="focuspoint"
       type="zope.schema.BytesLine">
  <form:widget
      type="z3c.form.browser.text.TextFieldWidget">
    <klass>text-widget pat-focuspoint-widget</klass>
  </form:widget>
  <title i18n:translate="">Image focus point</title>
  <required>false</required>
</field>
...
```