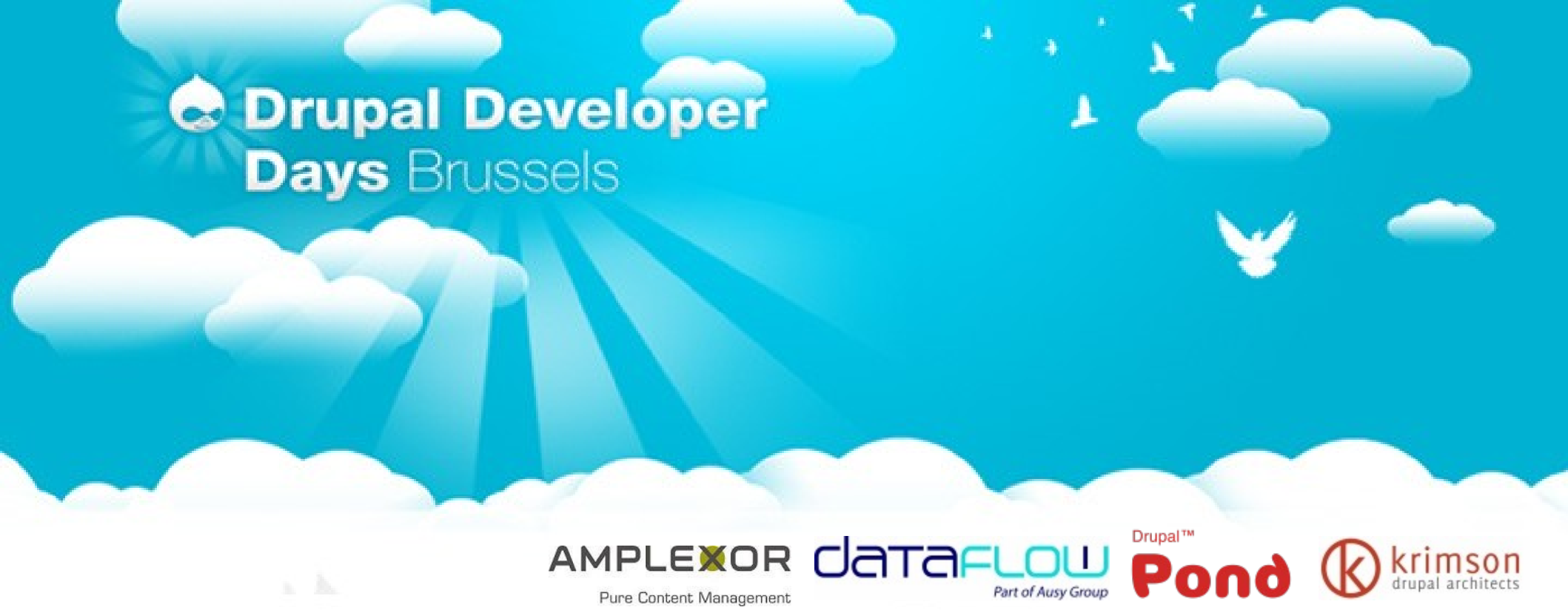

Leverage entities!

by Wolfgang Ziegler // fago

Who am I?

- Wolfgang Ziegler // fago
- Drupal developer from Vienna, Austria; working for epiqo

epiqo.

- Form API co-maintainer, Maintainer of

  Entity API, Rules, Content Access, Content Profile, Profile2, Auto nodetitle, Field-collection, RestWS, ...

- Co-founder and active member of Drupal-Austria http://drupal-austria.at

- http://wolfgangziegler.net

Overview

- Short introduction
- What are entities?
- How to develop with entities?
- How to provide a new entity type?

Entity API

- So this talk is about the
  - Drupal 7's entity API
  - and the Entity API module

http://drupal.org/drupal-7.0
http://drupal.org/project/entity

What are entities?

- Entities are the foundation for fields, but

Entity != Fieldable Entity

- While most entity types are fieldable, not all are. Fields are optional.
- Entity types defined in core:
  nodes, users, taxomony terms, vocabularies, comments, files.

If not fields, what else is it about?

Entities are required to be

- Loadable (via entity_load())
- Identifiable (what should be loaded?)

- That's it.
- It's not even required for an entity to live in the local database.

But there is more...

- But core provides more. There is support for
  - Looking up labels (via entity_label())
  - Looking up URIs (via entity_uri(), view only)
  - View modes – define multiple view modes
- And there are common hooks:

  hook_entity_load(), hook_entity_presave(), hook_entity_insert(), hook_entity_update(), hook_entity_delete(), hook_entity_prepare_view, hook_entity_view().
- And...

EntityFieldQuery

- API for querying entities
  - Supports conditions on entity properties as well as fields
  - Supports queries across multiple entity types (for fields).
  - Covers entity properties in the local database and querying fields in a single storage
- Extendable via hook_entity_query_alter().

Fields !

- Optionally, entity types are fieldable!
- The field API requires a numeric identifier ('id' key)
- Optionally, you may define bundles and revisions.
- Bundles define which fields are available.
  - Default bundle == entity type
  - Node types for nodes, Vocabularies for terms.

Entity API module...

- So far, this all is in Drupal 7.
- The Entity API module builds upon that to
  - ease dealing with any entity type
  - easy providing new entity type

    (optionally fieldable, exportable)

Working with entities...

- entity_load(), entity_save(), entity_create(), entity_delete()
- entity_view(), entity_access(), entity_label(), entity_uri()
- entity_id(), entity_extract_ids()
- entity_export(), entity_import()
- entity_get_info(), entity_get_property_info()

core

Entity property info // Metadata

- A hook defined by the Entity API module.
- Allows modules to describe entity properties (including fields).
- Properties have to be described by a set of defined data types ('text', 'date', 'user', ..)
- Includes human readable description, 'getter callback', optionally a setter, access permissions.

Property info example

```
// Add meta-data about the basic node properties.
$properties = &$info['node']['properties'];

$properties['title'] = array(
  'label'  => t("Title"),
  'type' => 'text',           // Default!
  'description' => t("The title of the node."),
  'setter callback' => 'entity_property_verbatim_set',
  'query callback' => 'entity_metadata_table_query',
  'required' => TRUE,
);
$properties['author'] = array(
  'label' => t("Author"),
  'type' => 'user',
  'description' => t("The author of the node."),
  'getter callback' => 'entity_metadata_node_get_properties',
  'setter callback' => 'entity_metadata_node_set_properties',
  'setter permission' => 'administer nodes',
  'required' => TRUE,
);
```

Entity metadata wrappers

- Some useful wrapper classes that ease dealing with entities and their properties.

```
$wrapper = entity_metadata_wrapper('node', $node);
$wrapper = entity_metadata_wrapper('node', $nid);

$wrapper->author->profile->field_name->value();
$wrapper->author->profile->field_name->set('New name');

$wrapper->language('de')->body->summary->value();

$wrapper->author->mail->access('edit') ? TRUE : FALSE;
$wrapper->author->roles->optionsList();

$wrapper->field_files[0]->description = 'The first file';
$wrapper->save();
$node = $wrapper->value();
```

How to provide a new entity type?

- Implement hook_entity_info().
- Specify your controller class.
  - Make use of the entity API controller to get full CRUD support, via entity_{CRUD}() out of the box.
  - Define your schema via hook_schema().
- Best practice:
  - add {entity_type}_load(), {entity_type}_create(), {entity_type}_delete(), {entity_type}_save(),

Entity CRUD API

→ **EntityAPIController :** For entity types stored in the local db.

- Invokes all usual CRUD related hooks for you. Document them using the documentation template (→ handbooks).
- Supports entities based on class, e.g. derived from the class “Entity”
- Having class eases
  - customizing labels, URIs, display or saving.
  - Is just useful:

```
$entity->entityType();
$entity->identifier();
$entity->view();
$entity->delete();
$entity->save();
$entity->label(), $entity->uri(), $entity->entityInfo(), ..
```

Entity CRUD API & Fields

- Define the entity type as fieldable and you are done (controller invokes FieldAPI for you).
- Provides entity_view() including fields.
- Supports you with managing bundles:
  - Add a separate entity type for bundles → Bundle entity (Profile type, Profile)
  - Specify the bundle entity type to be 'bundle of' another entity type (Profile type is bundle of Profile)
  - Again, Field API attachers are invoked for you.

Entity CRUD API: Example Profile2

```
/**
 * Implements hook_schema().
 */
function profile2_schema() {
  $schema['profile'] = array(
    'description' => 'Stores profile items.',
    'fields' => array(
      'pid' => array(
        'type' => 'serial',
        'not null' => TRUE,
        'description' => 'Primary Key: Unique profile item ID.',
      ),
      'type' => array(
        'description' => 'The {profile_type}.type of this profile.',
        'type' => 'varchar',
        'length' => 32,
        'not null' => TRUE,
        'default' => '',
      ),
.....
```

Entity CRUD API: Example Profile2

```
/**
 * Implements hook_entity_info().
 */
function profile2_entity_info() {
  return array(
    'profile2' => array(
      'label' => t('Profile'),
      'entity class' => 'Profile',                          // optional
      'controller class' => 'EntityAPIController',
      'base table' => 'profile',
      'fieldable' => TRUE,
      'entity keys' => array(
        'id' => 'pid',
        'bundle' => 'type',
      ),
      'label callback' => 'entity_class_label',       // optional
      'uri callback' => 'entity_class_uri',           // optional
      'module' => 'profile2',
    ),
    // ....
  );
}
```

Entity CRUD API: Example Profile2

```
/**
 * The class used for profile entities.
 */
class Profile extends Entity {

  public function __construct($values = array()) {
    parent::__construct($values, 'profile2');
  }

  public function defaultLabel() {
    return isset($this->label) ? $this->label : (isset($this->type) ? $this->type()-
>label : '');
  }

  public function defaultUri() {
    return array(
      'path' => 'user/' . $this->uid,
      'options' => array('fragment' => 'profile-' . $this->type),
    );
  }

  //...
}
```

Entity CRUD & Exportables

- Specify your entity type as exportable in hook_entity_info().
- Make use of machine names.
  - Your entity's main identifier will be the machine name, so entity_load('your_type', array('name')) works.
  - The 'id' key remains as numeric identifier, that is only used internally (storage, field API).

Entity CRUD & Exportables (2)

- Add entity_exportable_schema_fields() to your schema ('module', 'status').
- By default, hook_default_{entity_type}() will be invoked (+ the respective alter hook).
- Default entities will be synced to the database once modules are enabled / disabled / the cache is cleared.

  → You may rely on the db for sorting, querying..
  → You may rely on standard hooks (insert/update/..).

Example: Profile2 types.

```
/**
 * Implements hook_entity_info().
 */
function profile2_entity_info() {
   $return['profile2_type'] = array(
    'label' => t('Profile type'),
    'entity class' => 'ProfileType',
    'controller class' => 'EntityAPIController',
    'base table' => 'profile_type',
    'fieldable' => FALSE,
    'bundle of' => 'profile2',
    'exportable' => TRUE,
    'entity keys' => array(
      'id' => 'id',
      'name' => 'type',
      'label' => 'label',
    ),
    'module' => 'profile2',
    // ...


function profile2_get_types($type_name = NULL) {
  $types = entity_load('profile2_type', isset($type_name) ? array($type_name) : FALSE);
  return isset($type_name) ? reset($types) : $types;
}
```

Import / Export

- $export = entity_export($entity_type, $entity);
- $entity = entity_import($entity_type, $export);
- By default:
  - Export as JSON
    - Parse-able without security implications
    - Parsing JSON is simple.

Example profile type in code

```
/**
 * Implements hook_default_profile2_type().
 */
function recruiter_resume_default_profile2_type() {
  $items = array();
  $items['resume'] = entity_import('profile2_type', '{
    "userCategory" : false,
    "userView" : false,
    "type" : "resume",
    "label" : "Resume",
    "weight" : "0",
    "data" : { "use_page" : 1},
    "rdf_mapping" : []
  }');
  return $items;
}
```

Entity CRUD API: Module integrations

- The entity API helps you integrating with modules
  - Features integration for exportables
  - Very basic entity property info (read-only)
  - Basic Views integration based on the property info
  - Rules integration (events)
  - Token support for all entity properties via the “Entity tokens” module

Entity CRUD API: Property info

- Provide property info for your entity type, to
  - Get tokens for all properties, via entity tokens.
  - Get Views integration (as long the property has an equivalent column in the base table)
  - Obtain support of modules building upon it like
    - Rules
    - Search API
    - Restful Web Services (short: RestWS)

Entity CRUD API – Admin UI

- Enable the admin UI via hook_entity_info().
- Requires you to write an entity form +
  - {entity_type}_load()
  - An access callback for entity_access().
- Customize it by overriding the controller.
- UI suits very well for managing bundle entities.
  → Used by Profile2 for managing profile types.

Profile types

+ Add profile type + Import profile type

LABEL	STATUS	OPERATIONS					
Main profile (Machine name: main_profile)	Custom	edit	manage fields	manage display	clone	delete	export
Resume (Machine name: resume)	Default	edit	manage fields	manage display	clone		export

Entity Form // Best practices

- Use form id {entity_type}_form
- There is a wrapper function, putting
  - entity in $form_state[{entity_type}]
  - entity type in $form_state['entity_type']
- Update the entity in $form_state in your submit handlers and rebuild / finish.
- Use entity_form_submit_build_entity() to create {entity_type}_form_submit_build_{entity_type}, which submit handlers may re-use.

Entity API docs

→ in code entity.api.php

→ in the handbooks

http://drupal.org/node/878784

→ Learn from examples!

Learn from examples...

- The test module 'entity_test'.
- Profile2:
  - Fieldable profile entity
  - Exportable bundle entity (Profile type)
  - Admin UI implemented via the Entity API UI
  - Views / Rules / Features integration via Entity API
- Other modules using it
  Rules, Organic groups, Field-collection, Drupal commerce, Message, Rest web services, ..

So back to the start...

- What is an entity?

So back to the start...

- What is an entity?

A data unit.

So back to the start...

- What is an entity?

A data unit.

A data unit of which Drupal is aware of.
Stored anywhere.

Entity API for non-fieldable stuff? y?

- Save your time (by not writing CRUD functions)
- Benefit from a solution proved to work (tests!)
- Benefit from existing tools, e.g. (static) caching.
- Improve the DX
  - Get unified CRUD functions
  - Get unified hooks → Modularity.
  - Unified way to discover “data types”, their ids, labels, …
  - ...

Thanks!

Questions?