Entities

TABLE OF CONTENTS

Overview

The Entity API is used for manipulating entities (CRUD: create, read, update, delete). Entity validation has its own API (which could validate an Entity saved via REST, rather than a form, for example).

Entities come in two flavors: Content and Config(uration). The data storage mechanism moved from being field-centric in Drupal 7 to entity-centric in Drupal 8, 9 and 10. This implies that all fields attached to an entity share the same storage backend, making querying a lot easier. Entity types are registered with Drupal as plugins.

While Drupal supports custom entities, I haven't found a need for them in my projects. In most of my experience, just using real nodes (sometimes unpublished) has sufficed admirably.

Config entity types

They store configuration information e.g: views, imagestyles, roles, NodeType (which define node bundles).

- They are not revisionable
- They don’t have fields/are not fieldable.
- They don’t support entity translation interface(TranslatableInterface), but can still be translated using config’s translation API.

Content entity types

e.g. comment, user, taxonomy term, node and now media (bundles for media are file, image, audio, video, remote video etc.)

more at https://www.drupal.org/docs/drupal-apis/entity-api/introduction-to-entity-api-in-drupal-8

Query an entity by title and type

This example counts the number of entities of type `article` with the name `$name`. Note that access checking must be explicitly specified on content entity queries.

```
$query = $this->nodeStorage->getQuery()
  ->accessCheck(FALSE)
  ->condition('type', 'article')
  ->condition('title', $name)
  ->count();

$count_nodes = $query->execute();

if ($count_nodes == 0) {
```

Create an entity

To create a new entity object, use the entity_create. **NOTE** that this only creates an entity object and does not persist it.

```
$node = entity_create('node', array(
  'title' => 'New Article',
  'body' => 'Article body',
  'type' => 'article',
));
```

If you know what the entity class name (bundle type) is, you can use it directly.

```
$node = Node::create(array(
  'title' => 'New Article',
  'body' => 'Article body',
  'type' => 'article',
));
```

Save an entity

Entity save is done by calling the instance's save method.

```
$node->save();
```

Create article node entity with attached image

```
use Drupal\node\Entity\Node;

  $data = file_get_contents('https://www.drupal.org/files/druplicon-small.png');
  $file = file_save_data($data, 'public://druplicon.png', FILE_EXISTS_RENAME);

  $node = Node::create([
    'type'        => 'article',
    'title'       => 'A new article',
    'field_image' => [
      'target_id' => $file->id(),
      'alt' => 'Drupal',
      'title' => 'Drupal logo'
    ],
  ]);
  assert($node->isNew(), TRUE);
  $node->save();
  assert($node->isNew(), FALSE);
```

Update a node entity and add some terms

```
use Drupal\node\Entity\Node;
use Drupal\taxonomy\Entity\Term;

$node = Node::load(4);
$term1 = Term::load(1);
$term2 = Term::load(2);
$node->field_tags->setValue([$term1, $term2]);
$node->save();
```

Get the entity type and content type (or bundle type)

```
use Drupal\Core\Entity\EntityInterface;

function hook_entity_presave(EntityInterface $entity) {
  // getEntityTypeId() returns 'node'.
  // $entity->bundle() returns 'article'.
  if($entity->getEntityTypeId() == 'node' && $entity->bundle() == 'article') {
    // Do your stuff here
  }
}
```

Identify entities

from https://www.drupal.org/docs/drupal-apis/entity-api/working-with-the-entity-api

```
// Make sure that an object is an entity.
if ($entity instanceof \Drupal\Core\Entity\EntityInterface) {
}

// Make sure it's a content entity.
if ($entity instanceof \Drupal\Core\Entity\ContentEntityInterface) {
}
// or:
if ($entity->getEntityType()->getGroup() == 'content') {
}

// Get the entity type or the entity type ID.
$entity->getEntityType();
$entity->getEntityTypeId();

// Make sure it's a node.
if ($entity instanceof \Drupal\node\NodeInterface) {
}

// Using entityType() works better when the needed entity type is dynamic.
$needed_type = 'node';
if ($entity->getEntityTypeId() == $needed_type) {
}
```

Create a file entity

Note. Drupal is ok with files existing (for example in sites/default/files) without file entities but it probably makes more sense to have file entities connected with those files. When you create file entities, Drupal tracks the files in the tables: file_managed and file_usage. . You can also create media entities where bundle='file'.

Once the file already exists, I write the file entity. I do need a mime type

```
switch (strtoupper($type)){
  case 'PDF':
    $mimetype = 'application/pdf';
    break;
  case 'JPG':
    $mimetype = 'image/jpeg';
    break;
  case 'POW':
    $mimetype = 'application/vnd.ms-powerpoint';
    break;
}


// Create file entity.
$file = File::create();
$file->setFileUri($destination);
$file->setOwnerId(1);
$file->setMimeType($mimetype);
$file->setFileName($this->fileSystem->basename($final_destination));
$file->setPermanent();
$file->save();

$fid = $file->id();
return $fid;
```

Entity Validation

This looks quite interesting although I haven't had an opportunity to try it yet.

From Entity Validation API overview at https://www.drupal.org/node/2015613

The entity validation API should be used to validate values on a node, like any other content entity. While that was already possible to define validation constraints for base fields and for a field type, there was no API to add constraints to a specific field. This has been addressed.

```
/**
 * Implements hook_entity_bundle_field_info_alter().
 */
function mymodule_entity_bundle_field_info_alter(&$fields, \Drupal\Core\Entity\EntityTypeInterface $entity_type, $bundle) {
  if ($entity_type->id() == 'node' && !empty($fields['myfield'])) {
    $fields['myfield']->setPropertyConstraints('value', [
      'Range' => [
        'min' => 0,
        'max' => 32,
      ],
    ]);
  }
}
```

Custom validation constraints can also be defined. These links may be useful: ForumLeafConstraint/ and ForumLeafConstraintValidator

Also from https://drupalize.me/tutorial/entity-validation-api?p=2792 (you need a paid membership to read the whole tutorial): Drupal includes the Symfony Validator component, and provides an Entity Validation API to assist in validating the values of fields in an entity. By using the Entity Validation API you can ensure that you'r validation logic is applied to Entity CRUD operations regardless of how they are triggered. Whether editing an Entity via a Form API form, or creating a new Entity via the REST API, the same validation code will be used.

Resources

- Entity API on Drupal.org updated Jan 2021
- Introduction to Drupal Entity API from Drupal.org updated Sep 2022
- Drupal entity API cheat sheet Updated July 2018

- Drupal 8 Entity query API cheat sheet by Keith Dechant , Software Architect - updated February 2021
- Entity Validation API tutorial from https://drupalize.me (requires paid membership to read the whole tutorial) updated February 2022 at

---

Back to top

Page last modified: Apr 13 2023.

Edit this page on GitHub