BADCamp, October 24th 2015

Remote Entities: Past, Present & Future

Dave Bailey - steel-track
Colan Schwartz - colan

About Dave

- Drupal Architect for Autodesk Consumer Group
- Full-time Drupal Developer for 5 years
- Web Developer for over 7 years
- Drupal.org username - steel-track
- Contributed Modules: Advanced Page Expiration | Replicate Bean

AUTODESK®

About Colan

- colan on drupal.org
- Enterprise Web Architect specializing in Drupal
- Independent Contractor / Chapter Three
- Since 2006 (4.6!), contributed to 30+ modules
- (Co-)maintain ~20 modules, including:
  - Views
  - Context
  - Workbench Moderation

Example Project

Integrate Drupal with a REST API service that contains
tens of millions of data assets that are
updated every second by
millions of users.

Choosing a Solution

The solution for a data integration can be determined by three primary criteria:

- **consistency of the data**
- **size of the dataset**
- **maximum tolerance for stale data**

Common External Data Integration Approaches for Drupal

- Data Migration
- Non-Drupal PHP
- Client-Side

Data Migration Approach

Server-side Data migration at a standard interval
Ex: Feeds module at midnight

Server-Side Data migration at a non-standard interval
Ex: Migrate module upon trigger by a content admin.

Pros

- Easier debugging for edge cases and “dirty” data
- Performant (once it’s done)
- Native in Drupal (once it’s done)

Cons

- Stale data
- Monolithic data updates are risky
- Can take a very long time

Non-Drupal PHP Approach

PHP Library

Ex: Vendor provides a library that is loaded by Drupal and used to perform CRUD operations.

PHP Script

Ex: Custom PHP script is written and included in various templates as needed.

Pros

- No stale data
- Symfony integration with Drupal 8 makes non-Drupal PHP less scary

Cons

- Requires a lot of “glue” work (theme functions, form API, etc.)
- Tight Drupal integration is expensive
- Performance is dependent on external data source

Client-Side Approach

jQuery / AJAX

Ex: AJAX call fills in key areas of a page upon load.

Embedded AngularJS app

Ex: Drupal provides “skeleton” and some content while AngularJS widgets are embedded in key areas on the page.

Pros

- No stale data
- Extremely Performant

Cons

- Requires two theme systems
- Separate layer on top of Drupal
- Page can be empty on load
- Requires a wide skill-set on the team

But what if we could have native Drupal integration and no stale data?

Entities

Entities are a standard way to handle CRUD operations for data objects in Drupal.
Ex: Nodes, Users, Taxonomy Terms, Field Collections, Beans

Entities use a shared PHP interface, are fieldable, allow for multiple display modes, and can use Entity Field Queries (EFQ).

Core entity API + Entity API module + Remote Entity API + EFQ Views

Do entities have to live locally?

History of Remote Entities

- Larry Garfield's Remote Data in Drupal: Museums and the Web (2009)
  - Drupal 6
  - hook_menu() implementation with custom code for node loading
  - Future plans included per-field storage, but that's inefficient
- Florian Loretan's Remote entities in Drupal 7 (2012)
  - The entity API in Drupal Core
  - The Entity API contributed module
  - Repurposing EntityFieldQuery Views Backend to support remotes

Remote Entity API Approach

Remote Entity API module

Ex: Remote Entity API is leveraged to define custom entities with CRUD operations operating on a data storage external to Drupal and saving copies locally.

Pros

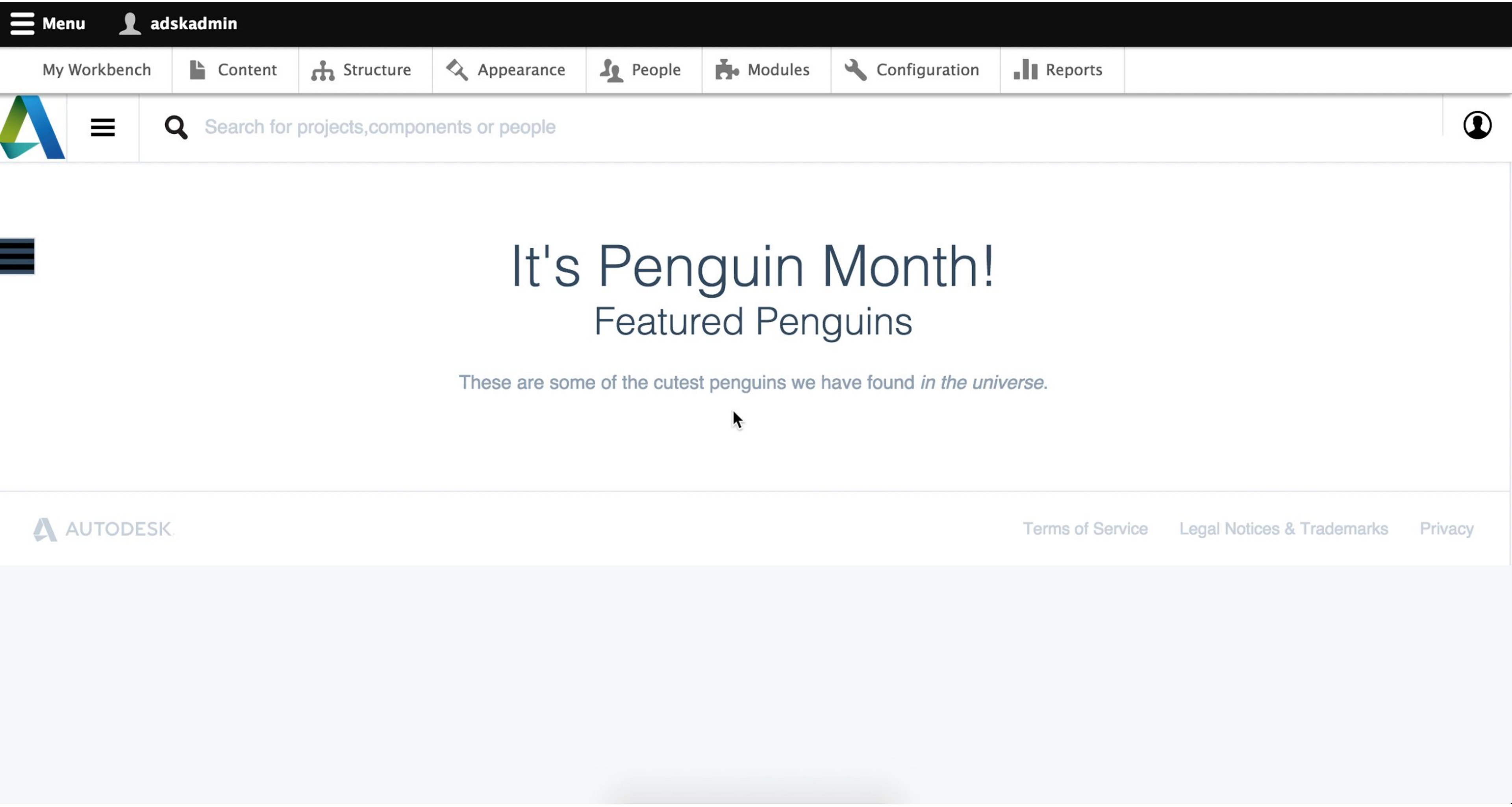
- Tightly integrated with Drupal
- ✓ Drupal Theming / Display modes
- ✓ Field API
- ✓ Views

Cons

- Saves data locally
- Performance is dependent on external data source
- Experimental

What was it like?

- How it started
- Potential client project
- Piecing together sparse docs
- Large holes
- Lots of debugging
- Missing DrupalCon
- Eventually
  - Loading
  - Views
- Wrote a partially comprehensive article
- Dave went further…



It's Penguin Month!

Featured Penguins

These are some of the cutest penguins we have found *in the universe*.

Issues with Remote Entity API

- Some great docs, but more are required
- Need one or more working examples
- Local copies shouldn’t be necessary
- Local saving confused with caching
- Dependency on Clients module
- Improve DX: some fixed snippets must be implemented in a custom module
- Make RemoteEntityQueries compatible with EntityFieldQueries to trivialize conversion

Performance Notes

It is like a database server. Slow queries, poor network, and inadequate throughput are still issues.

- Static Caching – Prevent redundant queries during a request
- Entity Caching – Subscribe to events from service to invalidate
- Big Pipe / Render Cache - Cache tags, cache contexts, placeholders
- Prevent requests wherever possible (ex: each page load)

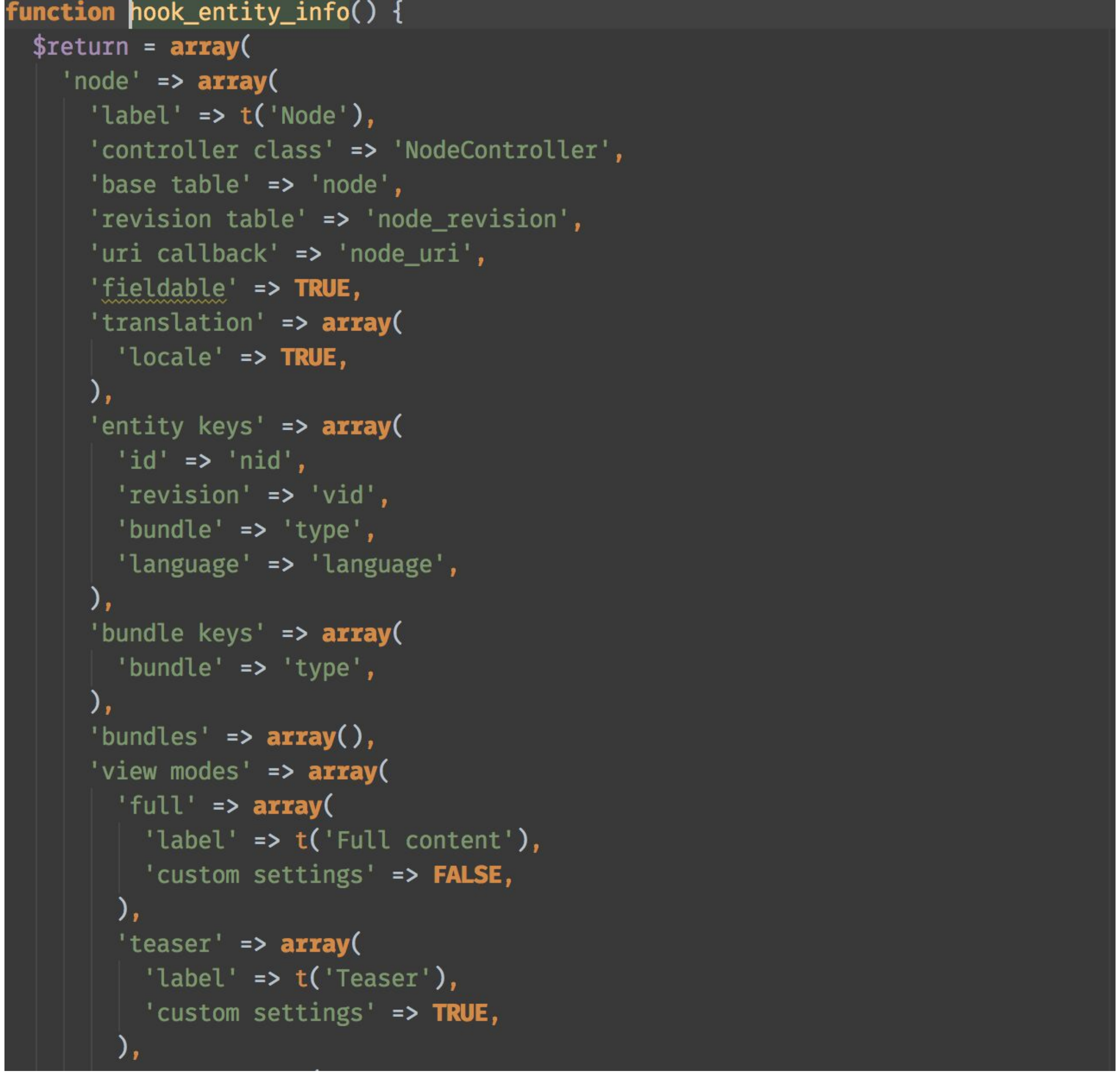
Why should Remote Entities be mainstream?

- Redefine the role of Drupal as an aggregator
- Fits into Drupal 8’s performance vision
- Reduce dependence on external data source by saving locally while pulling remotely

Steps to Implement

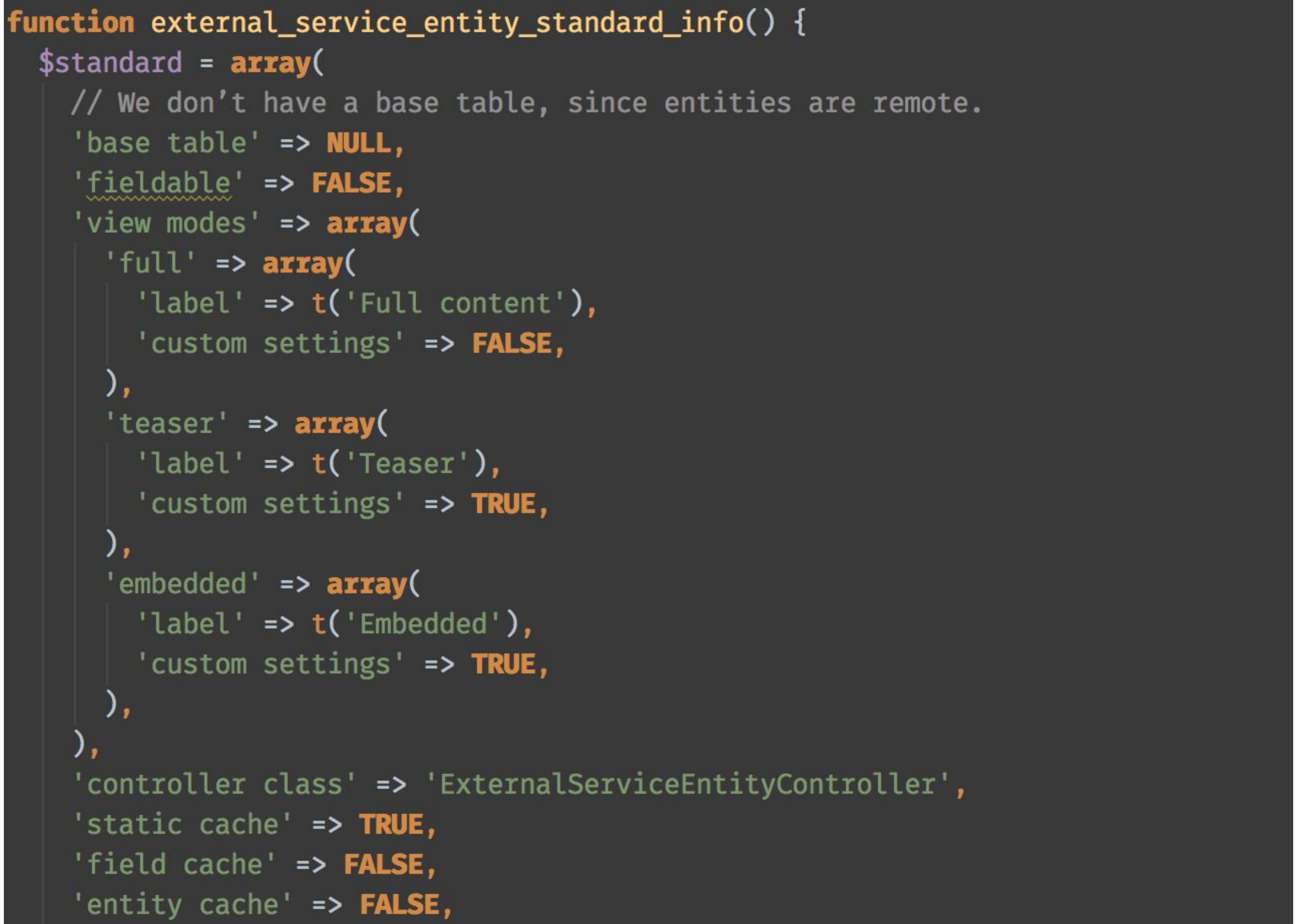
1. Create a custom entity controller
2. Define how to connect to the external service
3. Define how to query the external service
4. Define a custom entity
5. Define entity properties
6. Refine Drupal integration points
7. Implement a caching strategy

Defining a Custom Entity

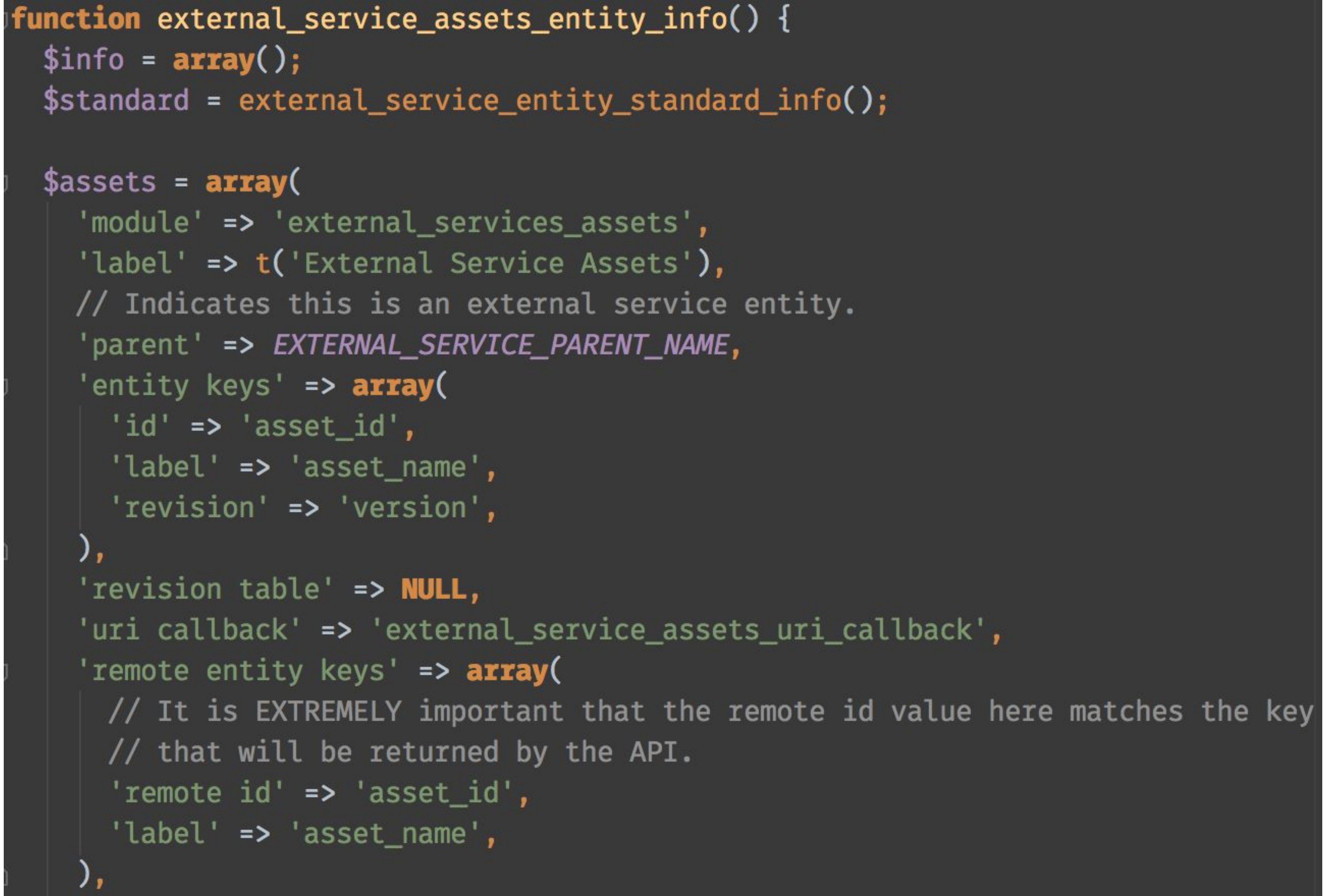


```
function hook_entity_info() {
  $return = array(
    'node' => array(
      'label' => t('Node'),
      'controller class' => 'NodeController',
      'base table' => 'node',
      'revision table' => 'node_revision',
      'uri callback' => 'node_uri',
      'fieldable' => TRUE,
      'translation' => array(
        'locale' => TRUE,
      ),
      'entity keys' => array(
        'id' => 'nid',
        'revision' => 'vid',
        'bundle' => 'type',
        'language' => 'language',
      ),
      'bundle keys' => array(
        'bundle' => 'type',
      ),
      'bundles' => array(),
      'view modes' => array(
        'full' => array(
          'label' => t('Full content'),
          'custom settings' => FALSE,
        ),
        'teaser' => array(
          'label' => t('Teaser'),
          'custom settings' => TRUE,
        ),
```

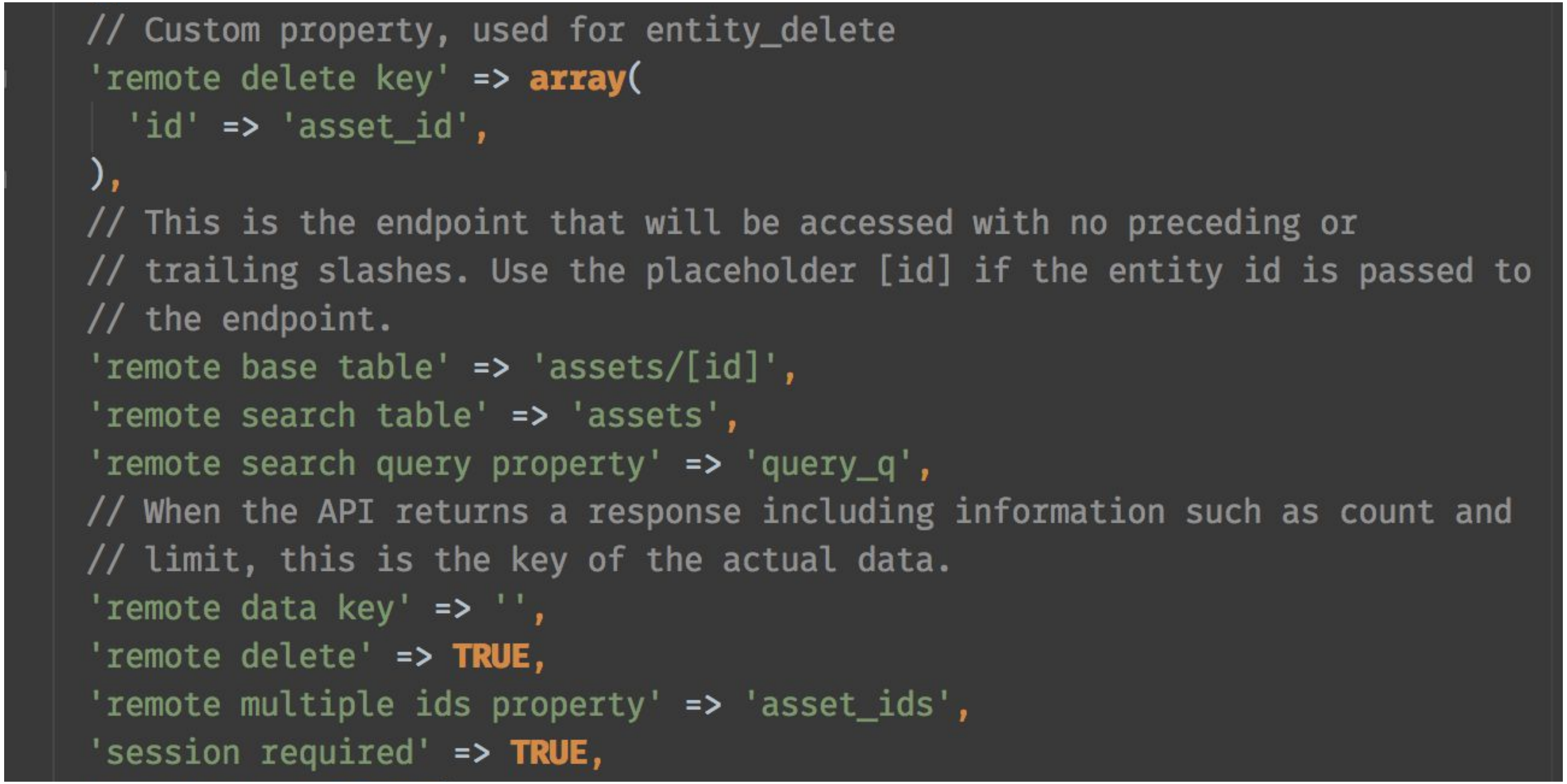
Defining a Custom Remote Entity



```php
function external_service_entity_standard_info() {
  $standard = array(
    // We don’t have a base table, since entities are remote.
    'base table' => NULL,
    'fieldable' => FALSE,
    'view modes' => array(
      'full' => array(
        'label' => t('Full content'),
        'custom settings' => FALSE,
      ),
      'teaser' => array(
        'label' => t('Teaser'),
        'custom settings' => TRUE,
      ),
      'embedded' => array(
        'label' => t('Embedded'),
        'custom settings' => TRUE,
      ),
    ),
    'controller class' => 'ExternalServiceEntityController',
    'static cache' => TRUE,
    'field cache' => FALSE,
    'entity cache' => FALSE,
```



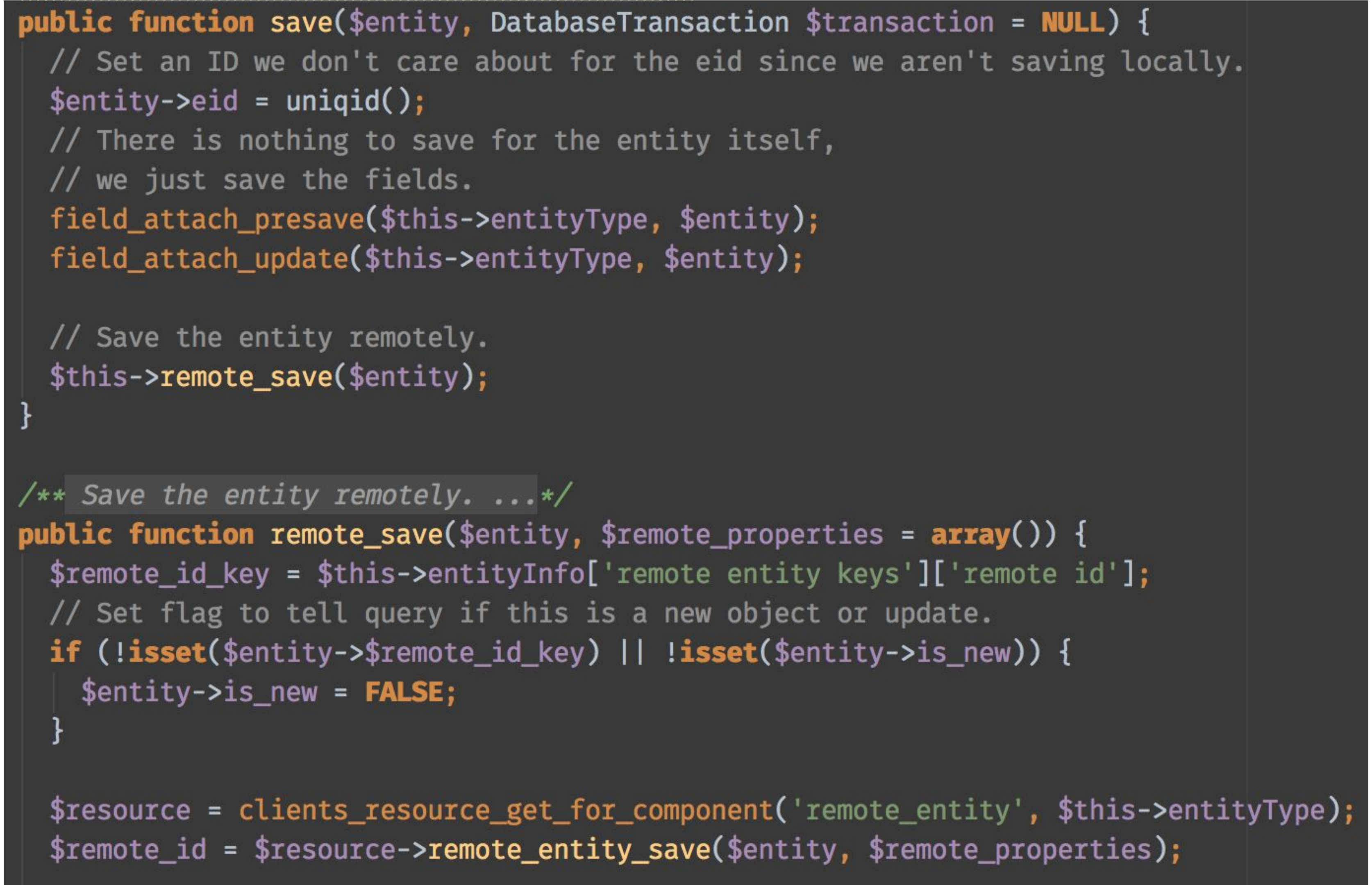
```
function external_service_assets_entity_info() {
  $info = array();
  $standard = external_service_entity_standard_info();

  $assets = array(
    'module' => 'external_services_assets',
    'label' => t('External Service Assets'),
    // Indicates this is an external service entity.
    'parent' => EXTERNAL_SERVICE_PARENT_NAME,
    'entity keys' => array(
      'id' => 'asset_id',
      'label' => 'asset_name',
      'revision' => 'version',
    ),
    'revision table' => NULL,
    'uri callback' => 'external_service_assets_uri_callback',
    'remote entity keys' => array(
      // It is EXTREMELY important that the remote id value here matches the key
      // that will be returned by the API.
      'remote id' => 'asset_id',
      'label' => 'asset_name',
    ),
```



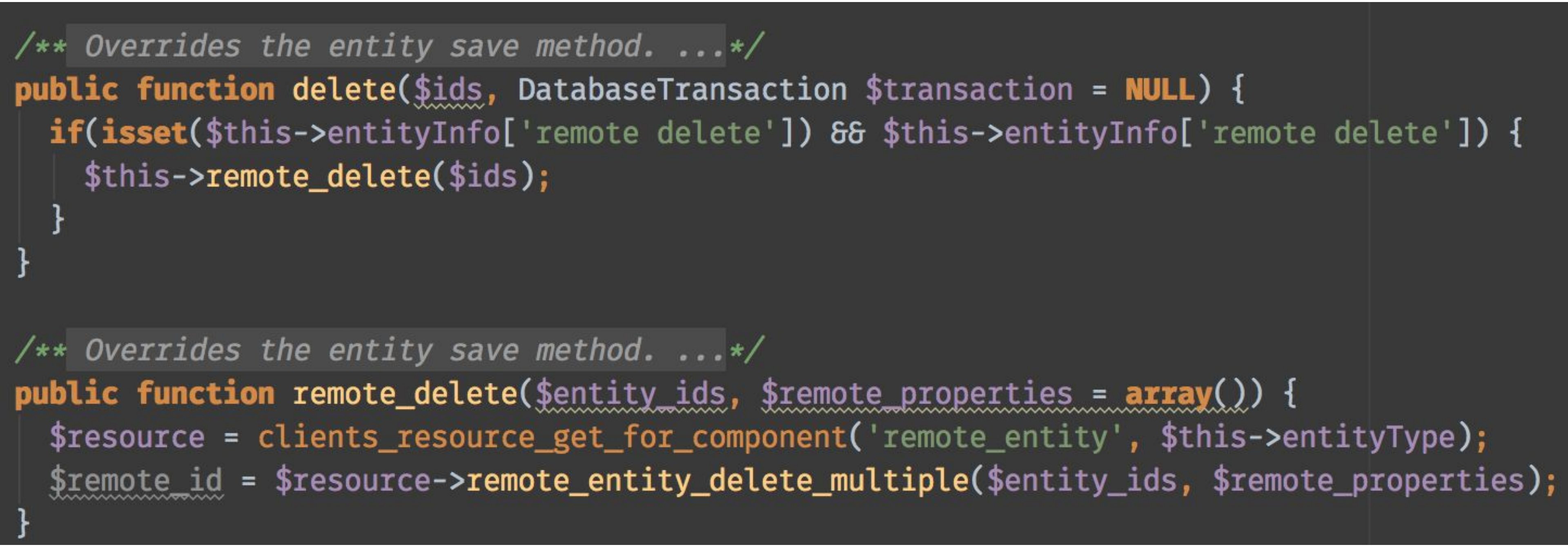
```
// Custom property, used for entity_delete
'remote delete key' => array(
  'id' => 'asset_id',
),
// This is the endpoint that will be accessed with no preceding or
// trailing slashes. Use the placeholder [id] if the entity id is passed to
// the endpoint.
'remote base table' => 'assets/[id]',
'remote search table' => 'assets',
'remote search query property' => 'query_q',
// When the API returns a response including information such as count and
// limit, this is the key of the actual data.
'remote data key' => '',
'remote delete' => TRUE,
'remote multiple ids property' => 'asset_ids',
'session required' => TRUE,
```

The Controller Class

The controller class implements the entity controller interface in order to define how CRUD operations are performed on the external service.

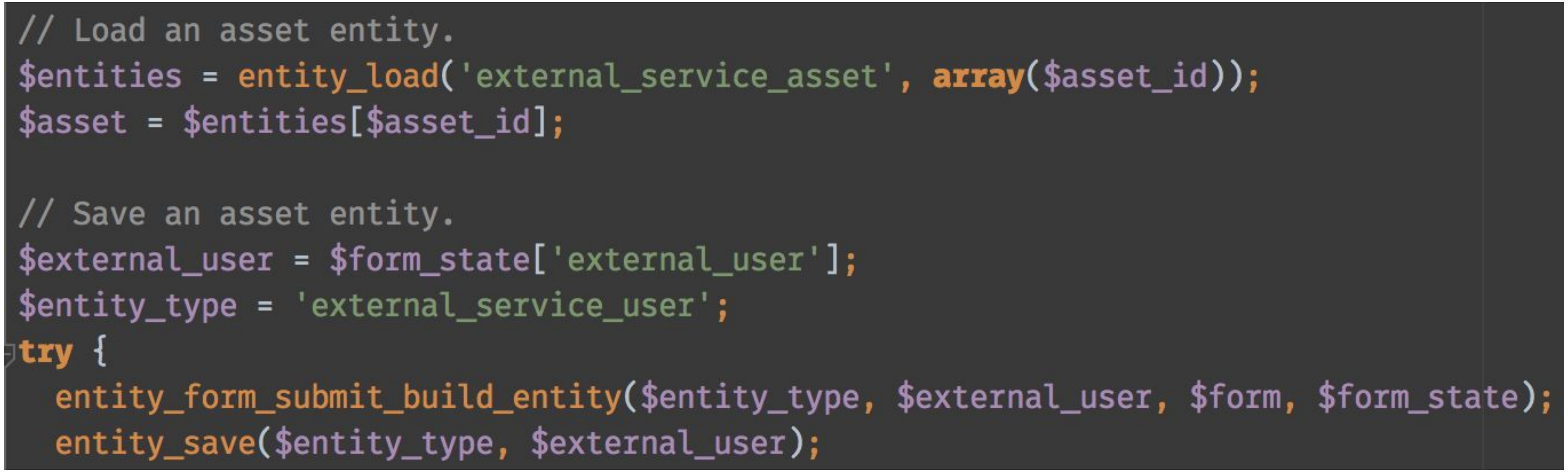


```php
public function save($entity, DatabaseTransaction $transaction = NULL) {
  // Set an ID we don't care about for the eid since we aren't saving locally.
  $entity->eid = uniqid();
  // There is nothing to save for the entity itself,
  // we just save the fields.
  field_attach_presave($this->entityType, $entity);
  field_attach_update($this->entityType, $entity);

  // Save the entity remotely.
  $this->remote_save($entity);
}

/** Save the entity remotely. ...*/
public function remote_save($entity, $remote_properties = array()) {
  $remote_id_key = $this->entityInfo['remote entity keys']['remote id'];
  // Set flag to tell query if this is a new object or update.
  if (!isset($entity->$remote_id_key) || !isset($entity->is_new)) {
    $entity->is_new = FALSE;
  }

  $resource = clients_resource_get_for_component('remote_entity', $this->entityType);
  $remote_id = $resource->remote_entity_save($entity, $remote_properties);
```



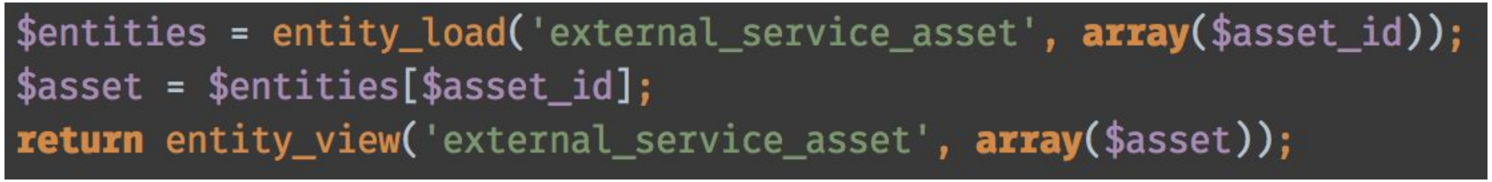
```
/** Overrides the entity save method. ...*/
public function delete($ids, DatabaseTransaction $transaction = NULL) {
  if(isset($this->entityInfo['remote delete']) && $this->entityInfo['remote delete']) {
    $this->remote_delete($ids);
  }
}

/** Overrides the entity save method. ...*/
public function remote_delete($entity_ids, $remote_properties = array()) {
  $resource = clients_resource_get_for_component('remote_entity', $this->entityType);
  $remote_id = $resource->remote_entity_delete_multiple($entity_ids, $remote_properties);
}
```

entity_load() and entity_save() work on remote data



```
// Load an asset entity.
$entities = entity_load('external_service_asset', array($asset_id));
$asset = $entities[$asset_id];

// Save an asset entity.
$external_user = $form_state['external_user'];
$entity_type = 'external_service_user';
try {
  entity_form_submit_build_entity($entity_type, $external_user, $form, $form_state);
  entity_save($entity_type, $external_user);
```

As well as entity_view()



```php
$entities = entity_load('external_service_asset', array($asset_id));
$asset = $entities[$asset_id];
return entity_view('external_service_asset', array($asset));
```

The Connection Class

The connection class defines how to connect to the external service, including credentials, error handling and response formatting. Associates endpoint resource with entity resource.

makeRequest() will ALWAYS be custom to your service

The Query Class

The query class defines how data is prepared to be sent to the external service. It converts all queries from Drupal into external service API formatted requests.

Query Class Notes

- EntityFieldQuery Views Backend
- Convert EntityFieldQuery to RemoteEntityQuery
- Implement buildFromEFQ() method
- EFQ Views will call this for remote entities
- Mostly generic, but still implementation specific
- See RemoteEntitySelectQuery::buildFromEFQ()
- On view creation
  - Use EFQ version of entity, not regular entity

The Query Class Converts Queries

Turn an object

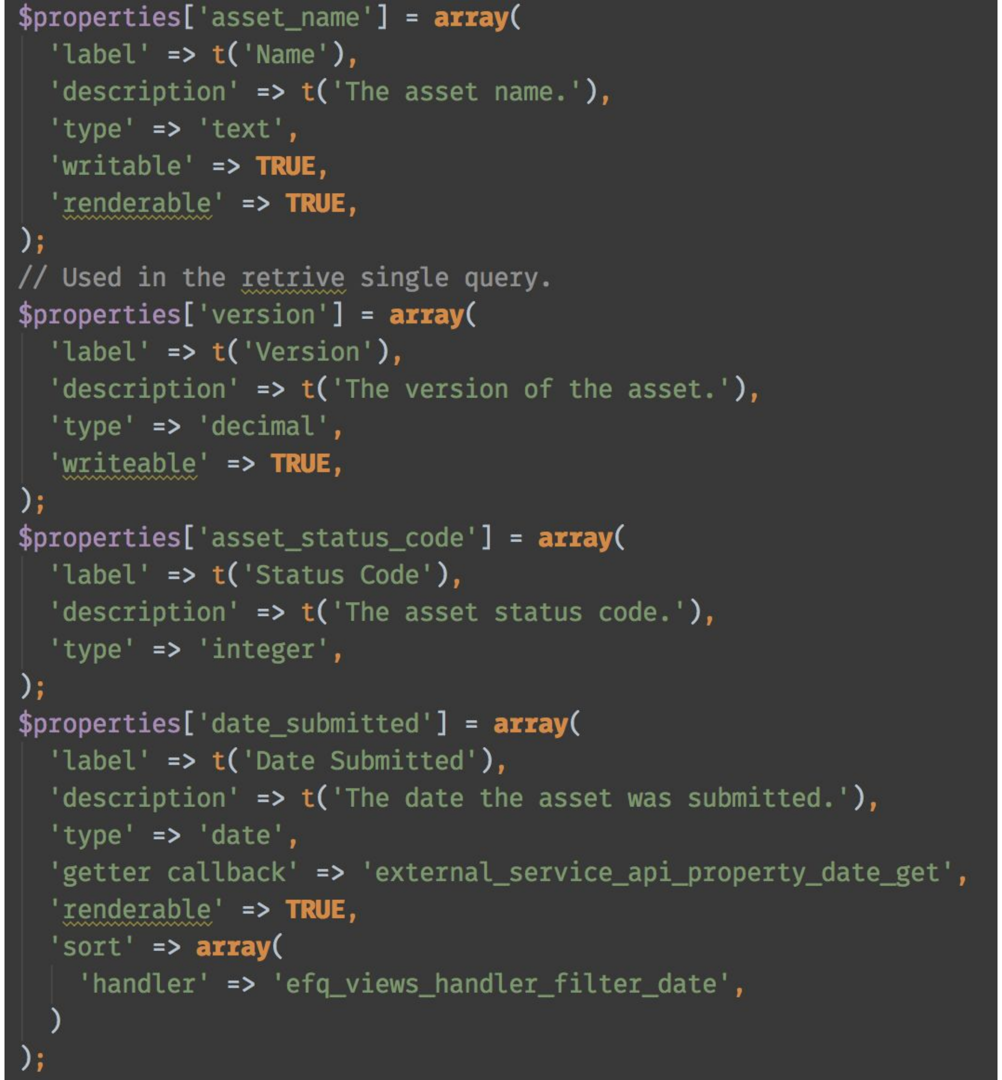
```
$object->propertyConditions = array(
  ‘time’ => ‘yesterday’,
  ‘user’ => 1234,
);
```

Into a REST API request

```
http://external.com/api/assets?time=yesterday&user=1234
```

Entity Properties

Entity API allows properties to be defined for an entity that describe the external data object to Drupal.

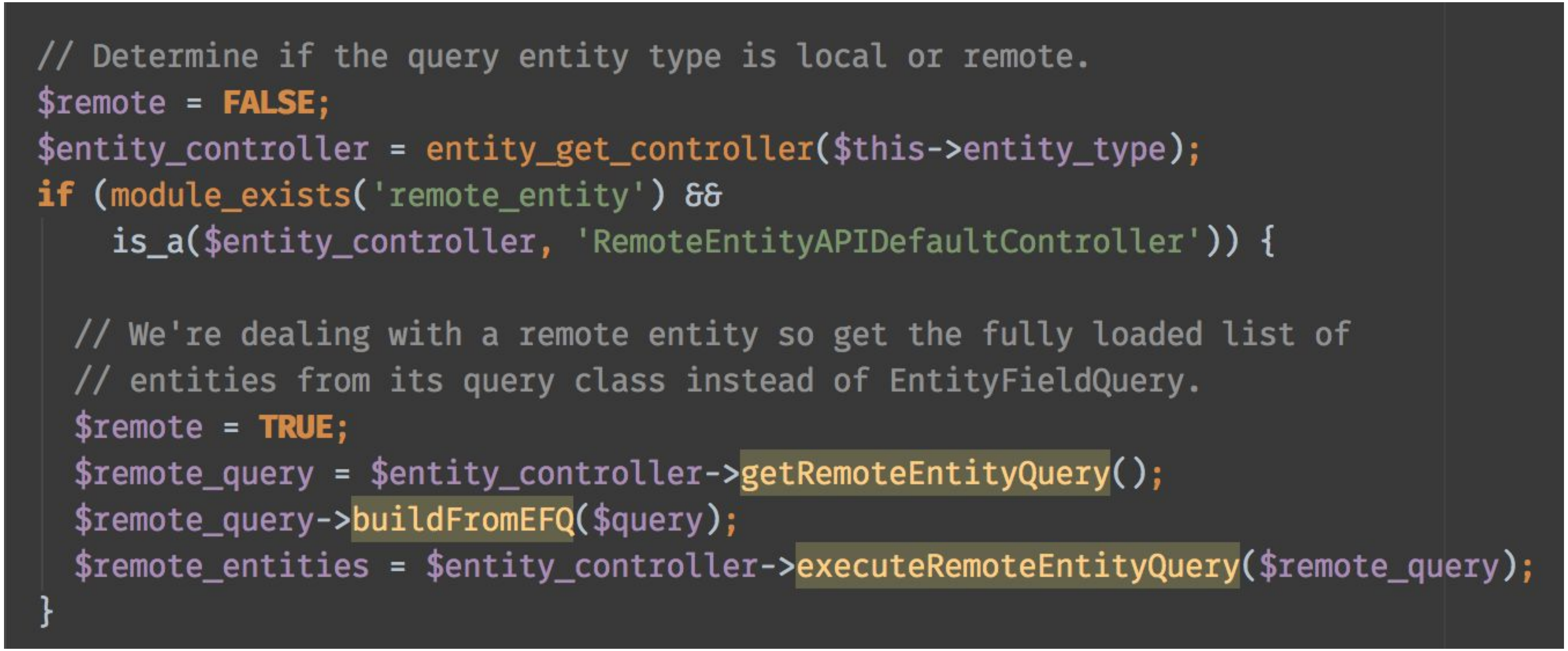


```
$properties['asset_name'] = array(
  'label' => t('Name'),
  'description' => t('The asset name.'),
  'type' => 'text',
  'writable' => TRUE,
  'renderable' => TRUE,
);
// Used in the retrive single query.
$properties['version'] = array(
  'label' => t('Version'),
  'description' => t('The version of the asset.'),
  'type' => 'decimal',
  'writeable' => TRUE,
);
$properties['asset_status_code'] = array(
  'label' => t('Status Code'),
  'description' => t('The asset status code.'),
  'type' => 'integer',
);
$properties['date_submitted'] = array(
  'label' => t('Date Submitted'),
  'description' => t('The date the asset was submitted.'),
  'type' => 'date',
  'getter callback' => 'external_service_api_property_date_get',
  'renderable' => TRUE,
  'sort' => array(
    'handler' => 'efq_views_handler_filter_date',
  )
);
```

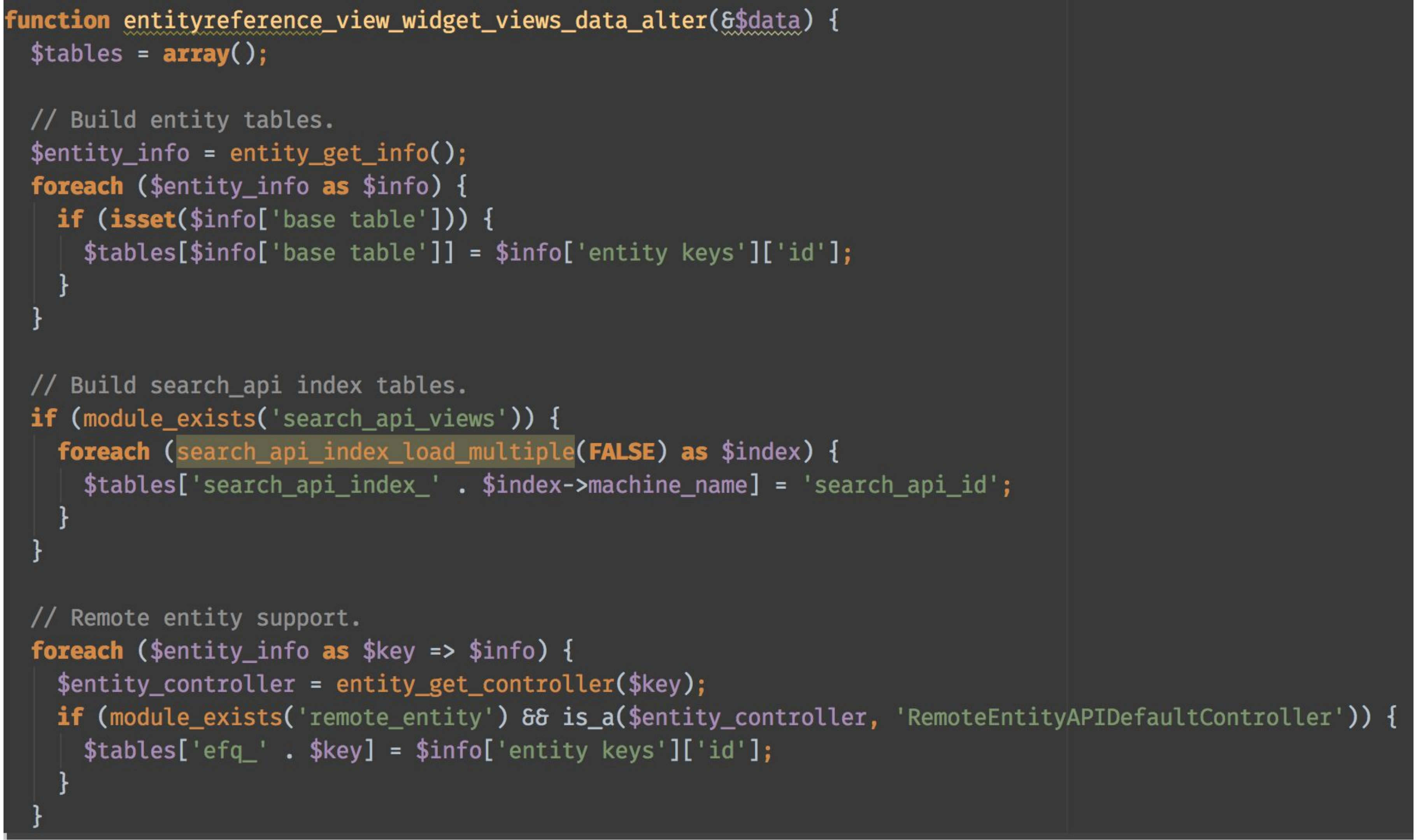
```php
$wrapper = entity_metadata_wrapper('external_service_asset', $asset);
$id = $asset->asset_id->value();
$sanitized_string = $asset->asset_name->value();
$formatted_date = $asset->date_submitted->value();
```

Drupal Module Integration

Most modules check if a base table is NULL and reject an integration. Views EFQ provides an example of the patch required.



```php
// Determine if the query entity type is local or remote.
$remote = FALSE;
$entity_controller = entity_get_controller($this->entity_type);
if (module_exists('remote_entity') &&
    is_a($entity_controller, 'RemoteEntityAPIDefaultController')) {

  // We're dealing with a remote entity so get the fully loaded list of
  // entities from its query class instead of EntityFieldQuery.
  $remote = TRUE;
  $remote_query = $entity_controller->getRemoteEntityQuery();
  $remote_query->buildFromEFQ($query);
  $remote_entities = $entity_controller->executeRemoteEntityQuery($remote_query);
}
```



```
function entityreference_view_widget_views_data_alter(&$data) {
  $tables = array();

  // Build entity tables.
  $entity_info = entity_get_info();
  foreach ($entity_info as $info) {
    if (isset($info['base table'])) {
      $tables[$info['base table']] = $info['entity keys']['id'];
    }
  }

  // Build search_api index tables.
  if (module_exists('search_api_views')) {
    foreach (search_api_index_load_multiple(FALSE) as $index) {
      $tables['search_api_index_' . $index->machine_name] = 'search_api_id';
    }
  }

  // Remote entity support.
  foreach ($entity_info as $key => $info) {
    $entity_controller = entity_get_controller($key);
    if (module_exists('remote_entity') && is_a($entity_controller, 'RemoteEntityAPIDefaultController')) {
      $tables['efq_' . $key] = $info['entity keys']['id'];
    }
  }
```

Drupal 8 Core

- Better entity API: no more hook_entity_info()!
- Everything is a class
- Fields are bound to an entity type (~~properties~~)
- Guzzle in now in core
- DrupalCon video: Entity Storage, the Drupal 8 Way
- *custom storage backend (connection, query)*
- *Conceptually, should be similar (and easier)*
- *Custom entity controller*

Drupal 8 Contrib

- ➔ Issue for Remote Entity API, but no movement
- ➔ Author won’t be working on it
- ➔ New External Entities module: code, no release
- ➔ EFQ Views: new branch, but no new code
    - Which module to support?

Drupal 7 Resources

- Blog Article by Colan Schwartz
  - [Integrating remote data into Drupal 7 and exposing it to Views]
- DrupalCon Barcelona video by Dave Bailey
  - [Remote Entities: Standardizing External Integrations in Drupal]
- Module by Joachim Noreiko
  - [MS Dynamics Client Connection reference implementation]
- Book by Sammy Spets
  - [Programming Drupal 7 Entities]

Where do we go next?

Thoughts?

Feedback?

Q & A?

Turn off the recording