Views for Developers

(and data geeks)

Bec White
white@palantir.net
DrupalCamp WI, July 2010

What is Views?

"a tool for making lists of stuff"

Views describes the data in Drupal, and then layers filtering and display on top.

Developing with Views:

- Export Views
- Call Views in your code
- Providing data to Views
- Plugins and handlers

at this point you should be building a custom module.

hook_views_api()

```
function example_views_api() {
  return array(
    'api' => 2,
    'path' => drupal_get_path(...),
  );
}
```

- lives in example.module
- tells Views to look for two includes:
  - example.views.inc
  - example.views_default.inc

this tells Views that your module uses the views api

Exporting Views

(you should already be doing this)

for version control, moving Views from dev to production (or site to site)

if you just want to export views and keep them in code, but you're not developing a module for any other piece of functionality, use Features. The Features interface lets you check off one or more Views, and then download a custom feature module that provides those views.

hook_views_default_views()

```
function example_views_default_views() {
  $views = array();

  $view = new view;
  $view->name = 'example_view';
  $view->description = 'My list of stuff.';
  $view->base_table = 'node';
  $handler = $view->new_display('default', ... );
  $handler->override_option('fields', array( ... ));
  ...
  $views[$view->name] = $view;

  return $views;
}
```

paste in output from views 'export' tab -- OR use views bulk export module. basically, builds view configuration objects line by line, and returns an array of views.

OR, like I said, use features

Calling Views from code

please, please don't do this "in the database"

What do you want from the View?

- Rendered block of HTML
- `views_embed_view('view_name', 'default')`
- Array of View results
- `views_get_view_result('view_name', 'default')`
-
- The View object
- `views_get_view('view_name')`

Views as a data model

Views uses hook_views_data() implementations to describe Drupal data

also, many of the places you want to put rendered Views are already handled by Views displays--which is the actual point of display plugins.

if a display plugin isn't adequate, check out the code in views_embed_view() (lives in views.module). This function is a very simple way to render the results of a view, and is supposed to be an example if you want to do something more complicated.

Views as a query builder

takes the Views data model and builds SQL

Views as a display layer

"squeezes results through the theme layer"

Extending Views

- Describe your data to Views
- `hook_views_data()`
- Provide handlers for your data
- `hook_views_handlers()`
-
- Manipulate View results for output
- `hook_views_plugins()`

Each of these returns a configuration array of some sort

ie, "this is a list of the handlers I provide" or "this is a list of the data I provide"

Handlers vs. Plugins

handlers interpret data; plugins deal with results

crell does a talk with this same title. he usually talks more about plugins than handlers. my experience with views is mostly from the other direction--piping data into views--so I have more experience writing handlers.

when you describe a table in the database to Views, you tell views what handlers to use for fields, filters, arguments, and sorts

plugins deal with the results of the view, after a query has run

Plugins

- Display plugins - where should the results of a View appear?
  - on a page, in a block, attached to another displa
  - Views Attach - http://drupal.org/project/views_attach
- Style plugins - change the "shape" of the output
  - lists, tables, grids
  - calendars, map layers, tag clouds
- Row plugins - change the "shape" of data in an individual row
  - generally "node" or "fields"

hook_views_plugins()

```
function tablegroup_views_plugins() {
  return array(
    'module' => 'tablegroup',
    'style' => array(
      'tablegroup' => array(
        'title' => t('Grouped Table'),
        'help' => t('Displays results as cells in a table...'),
        'handler' => 'tablegroup_plugin_style_tablegroup',
        'theme' => 'tablegroup_view_tablegroup',
        'uses row plugin' => TRUE,
        'uses fields' => TRUE,
        'uses options' => TRUE,
        'type' => 'normal',
      ),
    ),
    'display' => array ( ... ), // looks similar
  );
}
```

these are all themeable

you shouldn't have to write a display plugin. check out the views attach module, which provides two extra display plugins -- "profile" and "node content" which let you embed views on user pages and on nodes.

the results of a view come through as an array. a style plugin can turn this array into whatever. an array makes a lot of sense as a list, but...

'argument' styles

choice between node ("render a node") vs. fields ("choose specific pieces of data to display")

styles may "use row plugin" or not; rows may "use fields" or not

documented in Views' Advanced Help

see also views/includes/plugins.inc

tablegroup_plugin_style_tablegroup.inc

```
class tablegroup_plugin_style_tablegroup extends views_plugin_style
{
  function option_definition() {
    $options = parent::option_definition();
    $options['row_grouping'] = array('default' => NULL);
    return $options;
  }

  function options_form(&$form, &$form_state) {
    parent::options_form($form, $form_state);
    $form['row_grouping'] = array( ... );
  }

  function render() {
    $output = 'some HTML';
    return $output;
  }

}
```

builds off the base views plugin style

views is OO, and a lot of the time you only need to override one method

Handlers

- fields
- filters
- sorts
- arguments
- relationships

and even

- joins
- default arguments

handlers adapt data in the back end for use with views fields, filters, arguments, and sorting

you declare these in hook_views_data()

there are a LOT of handlers in views core. generally it is a good idea to reuse these.

if you have complex data, or need to do custom formulas, or take advantage of database-specific features, you might want to consider writing a handler.

when you write a handler, you are usually writing database-specific code. you are not doing this on purpose, but really: do you use postgres? do you know the ansi sql spec?

hook_views_data()

- Use it if you have custom data (as database tables)
- OR use the Data module, which does it for you
  - http://drupal.org/project/data

Use hook_views_data() to describe your data model.

- "Base tables" represent core objects
  - node, user, comment
- other tables may join to base tables -- and add more stuff to what Views knows about a core object
  - CCK adds to nodes (fields, filters, arguments, sorts)
  - Profile module adds to users

hook_views_data() (continued)

```
function node_views_data() {
  $data['node']['title'] = array(
    'title' => t('Title'),
    'field' =>
      array('handler' => 'views_handler_field_node'),
    'sort' =>
      array('handler' => 'views_handler_sort'),
    'filter' =>
      array('handler' => 'views_handler_filter_string'),
    'argument' =>
      array('handler' => 'views_handler_argument_string'),
  );
  return $data;
}
```

this doesn't include several important parts

but: node is a core drupal object

node is also a base table in views

a filter handler

```
class views_handler_filter_string extends views_handler_filter {
  function option_definition() {
    $options = parent::option_definition();
    $options['case'] = array('default' => TRUE);
    return $options;
  }

  function options_form(&$form, &$form_state) {
    parent::options_form($form, $form_state);
    $form['case'] = array( ... );
  }

  function query() {
    $this->ensure_my_table();
    $field = "$this->table_alias.$this->real_field";
    $this->query->add_where($this->options['group'],
                            "$field = '%s'", $this->value);
  }
}
```

this is an approximate excerpt from views code

we always extend views_handler_x, in this case views_handler_filter, since there are LOTS of things that it handles--for filters, it will build the exposed form, deal with other options (note call to parent::option_definition(), etc), also little things like the 'admin summary' -- the word that shows up next to the filter in the views UI

Views documentation is accessible via Advanced Help

or in the 'help' directory of the Views module

Views 3 preview...

"area" plugins*, pager plugins*, exposed form plugins*, query plugins

Resources

- http://drupal.org/project/views
- http://drupal.org/project/views_attach
- http://drupal.org/project/tablegroup
- http://drupal.org/project/data

- Views API documentation (sort of):
  - http://views.doc.logrus.com/
  - in the module: views/help

- #drupal-views in IRC

* = themeable

area plugins = put something other than "text" into the view header and footer

pager plugins = page by #items (current), page by amount of time (1 month = 1 page), page by some other argument value (letter?)

query plugins = use a non-sql backend -- ie, describe the data provided by an external api (flickr), and then use views to make api queries

or nosql databases, or xml

AND MORE