csg
custom simple groupware framework

Author: Patrick Pliessnig
contact/support: google groups -> simple groupware

Introduction

When writing scripts in php to manipulate assets and folders, I like to focus as much as possible on the user problem domain. I do not want to be bothered by the complexity of php programming. The aim of csg is to ease the task of writing scripts for sgs. This is a crucial point, when I need to have a new and stable function ready for the next day. And after all, I am not a php programmer. My mind is somewhere else.

What is csg

It is a set of class definitions in php to manipulate sgs objects like assets, folders, schemas, etc. For example to have access to the features of a schema like the sql query or the fields, in a php script you can simply write

```
$s = new csgSchema(folder,view);
```

If you then want to know the field names available for this schema, you can get an array of field names by writing

```
$fn = $s -> field_names;
```

This comes quite handy. 1St your script becomes more readable, 2nd you keep your schema definitions like sql query all in the well known places like module.xml and tree diff-schema-definitions. No need to distribute sql queries wherever you want access to asset data. 3rd all security options of sgs are respected.

Another way to look at csg is to see it as a wrapper around sgs api. In its current state csg is a wrapper around the ajax api and is limited by its functionality, but can be extended of course. The way of working with csg is similar to click 'n point in the browser, but with writing php commands. To work well with csg you need to know what views, folder, assets and filters are.

What can you do with csg

You can write scripts for administration to run manually and custom functions for your users that they can use without even knowing what happens behind the scene. The scripts can be executed directly in the console or via

http://<your-server>/bin/index.php?csg=myscript... (other arguments might come here)

you can also use csg in combination with validation, store-methods, restore-methods, trigger, etc.

Your contribution

You can contribute by proposing general functionality and patterns to integrate into csg. Functionality should a) rely somewhat on sgs api in order to be easy to integrate and not to duplicate existing functionality b) integrate in some way to the overall object oriented approach of csg and c) be useful for the community.

If these criterias are met, I can integrate the proposed functionality or someone else can do it.

Integration of new functionality needs to pass my test of conformance before publishing.

Status of the framework

It is a work in progress. The functions I use in my environment are of course well tested, other functions not so well or not at all. As sgs lacks a little bit of technical documentation, it is rather time consuming to test everything. I you want to use a function that is not thoroughly tested, I am here to help.

Use at your own risk!

If you want stability, wait till stability is announced

If you want to help testing for your needs, you are welcome

Never process your scripts in a production environment without proper testing! With this kind of scripting it is easy to delete a bunch of data with one line of code.

Version

Version 0.1

Do not consider the interface to be stable.

Requirements

I have written the csg framework with php 5.3. it should also work well with higher versions.

Installation

Install csg

1. Move all the php files prefixes with 'csg' either to `../ext/core/classes/` or `../custom/core/classes/`
2. edit bin/index.php and replace

   ```
   folder_process_session_request();
   ```

with

```
csg::process_session_request();
folder_process_session_request();
```

(this way your script – if requested – will be processed before standard sgs processing occurs)

install your custom script

Default location of your custom script is `../custom/ext/lib/`. Simply move your script file (a filename with '.php' extension) to this folder (other locations can be configured as well).

Console

You can write your scripts and test them directly in sgs console. To inspect the state of an object `$o` for debugging, it is usually a good idea to use `print_r($o);`

to process your script in the console you can write:

```
$_REQUEST["csg"] = "myscript";
csg::process_session_request();
```

Calling your script with url

If the filename of your script is myscript.php, you can process it by calling

```
http://<your-server>/bin/index.php?csg=myscript
```

you can add other url parameters too like &folder=101&view=display etc...

you can change the resource name `csg` to you own resource name by using the command

```
csg::process_session_request($resource = 'myresource');
```

in the `bin/index.php` file and then process the script by calling

```
http://<your-server>/bin/index.php?myresource=myscript
```

Tutorial

1. After Installation you can directly write the example in the console. If you prefer writing a script myscript.php, you can process the script from the console with the commands

   ```
   $_REQUEST["csg"] = "myscript";
   csg::process_session_request();
   ```

2. now we want to list all the assets in the workspace/demo/contacts folder. Check the Id of the folder. In my case it is 801:

   first we create a schema variable:

   ```
   $demo_contacts = new csgSchema (
   ```

```
   $folder_id = 801, $view_name = "display");
```

2nd we create a folder variable out of the schema

```
$folder = $demo_contacts -> get_folder();
```

now we want to print a list of the assets on the screen. We do this with

```
print_ r( $folder -> assets );
```

it should be an array of csgAsset objects. The keys of the array are the ids of the assets.

Take out an asset of your choice. For example

```
$asset = $folder -> assets[101];
```

now print the lastname of the contact on the screen. You can do this with

```
print_r ( $asset->lastname );
```

now we want to change the adress of asset 101 to London, Baker Street 10

we start by creating a schema object for editing

```
$edit_contact = new csgSchema ($folder_id = 801, $view_name
   = "edit");
```

we load the asset of Doe John in a variable

```
$doe_john = csgAsset::get_from_db
   ($schema = $edit_contact, $id = 101);
```

we set his address

```
$doe_john -> street = “Baker Street 10”;
```

```
$doe_john -> city = “London”;
```

```
$doe_john -> country = “GB”;
```

finally, we need to update the asset in the database, we do this with

```
$doe_john -> update();
```

now you can inspect the result directly in sgs.

Reference

Abstract Class csg:

Represent the csg environment.

Const

default_script_library, default_resource_name

Static features

- process_session_request() // see introduction
- get_script_directory(), get_resource_name() // returns a string

Abstract Class csgScript:

Represent the script beeing processed.

Static features

- exists_argument($name) // returns boolean
- get_argument($name)
  // returns an csgScriptArgument object of name $name and corresponding value
- get_arguments()
  // returns an array of csgScriptArguments passed to the script. The array is indexed by the argument name.

Class csgScriptArgument

Instances represent an argument of the script beeing processed. Arguments are usually passed to the script by url parameters like ?folder=101&view=display (folder is the argument name, 101 is the argument value). Command line arguments are currently not implemented.

features

- name, value
  // properties of the object

example:

```
$my_argument = csgScript::argument("folder");
echo $my_argument -> value;
```

Class csgSchema

Instances represent the schema defined by a folder and a view. This includes the sql query and the fields for the view. csgSchema is central for sgs. So it is for csg. The folder_id can be an ID or a path like /Workspace/Demo/Contacts. The view is given as a name.

Create

- new csgSchema($folder_id, $view_name = "display")

features

- folder_id
  // property. the '/Workspace/Demo/Contacts' type of folder is not tested

- view_name // property
- field_names // property
- get_folder ($autorefresh = true)
  // returns a new csgFolder object for the schema.

Class csgAssetData

Instances represent a list of data fields. A data field is an array of the form array(field_name => value). a csgAssetData object is not related to a csgSchema. In sgs this is analog to what is commonly labeled as a $row.

Create

- new csgAssetData (array $fields = array())
  // a field is an array(field_name => value). a csgAssetData object can be initially empty

Features

- add_field ($name, $value) // adds a field to the object
- remove_field ($name) // removes a field from the object
- exists_field ($name) // returns true if field exists, false otherwise
- field_names // property, returns an array of the field names
- fields
  // given an object $data of type csgAssetData, the value of a field 'field_name' can be evaluated with: `$data -> field_name;` . Similarly the value of the field can be set with `$data -> fieldname = value;`

Class csgAsset

Instances represent assets in sgs. They are similar to csgAssetData objects, but are always related to a schema.

Create

- new csgAsset (csgSchema $schema, csgAssetData $data = null)

static features

- get_from_db ($schema, $id)
  // creates an instance of csgAsset and returns it initialized from database
- get_from_trigger_params(), get_from_rowfilter_params(), get_from_rowvalidate_params(), get_from_store_method_params(), get_from_restore_method_params(),
  // need implementation

features

- field_names
  // property, returns an array of the field names
- fields: fields can be set and evaluated similar to csgAssetData objects (see above). The id, folder and view cannot be changed manually
- insert_as_new()
  // inserts itself into the database. Creates a new idea
- update()
  // updates itself to the database. Needs a valid id
- validate()
  // validates itself. Not tested
- delete()
  // moves itself to the trash bin. Needs a valid id. Not tested
- purge()
  // deletes itself from the database. Needs a valid id. Not tested
- move($target_folder)
  // moves itself to the target_folder. Not tested
- copy($target_folder)
  // copy a clone of itself to the target_folder. Not tested
- is_success_last_operation()
  // returns true if the last operation was successful, false otherweise
- error_msg_last_operation()
  // returns a sgs error message, if the last operation was not successful

Class csgFilter

Instances represent a filter in the same way as the filters used in views.

Const

like, not_like, starts_with, equal, not_equal, less_than, greater_than, one_of
// valid operators

Create

- New csgFilter ($field_name, $operator, $search_string)

Features

- field_name // property
- operator // property

- search_string // property

Example: $myfilter = new csgFilter ('name', csgFilter::like , 'Faruq');

Class csgView

Instances represent the view of a folder

Create

- Get a view from a csgFolder object with `$myview = $folder -> view`
- add_filter (csgFilter $filter, $name = null)
  // adds a filter to the view with an optional name
- remove_filter ($name)
  // remove a filter with name $name
- reset_filter() // deletes all filters
- set_filter_on, set_filter_off
  // activates, deactivates filter handling
- field_names // property, returns a list of the field names
- schema // property, returns a schema
- filters // property, returns an array of the filters
- filter_is_activated // property, returns true if filter handling is activated

Class csgFolder

Instances represent a folder of a schema

Create

- Get a folder from a csgSchema object with `get_folder ( $autorefresh = true, $activate_filter = false )`
  // $folder = $schema -> get_folder ();
- view // property, returns a csgView object
- assets // property, returns an array of csgAsset objects
- refresh() // refreshes the asset list in the folder
- is_autorefresh() // returns true if refreshing ist automatic
- set_autorefresh ($mode = true) // sets autorefresh mode
- asset($id) // returns an csgAsset object with id
- asset_array($index_field = null)
  // returns an array of assets indexed by $index_field. $index_field must be unique. If

$index_field is null, the array is indexed by id

- asset_array_asc($index_field)
  // same as asset_array but in ascending order
- asset_array_desc($index_field)
  // same as asset_array but in descending order
- delete_asset($id)
  // moves asset to trash. Needs testing
- delete_assets (array $ids)
  // moves all the assets to trash. Needs testing
- delete_all_assets()
  // moves all assets in the view to trash. Needs testing
- purge_asset($id)
  // deletes asset from database
- purge_all_assets()
  // deletes all assets in view from database. Needs testing
- cut_paste_asset($id, $target_folder)
  // moves an asset to $target_folder. Needs testing
- copy_paste_asset($id, $target_folder)
  // copy an asset to $target_folder. Needs testing