Extend BuddyPress - Part One.
Exploring the Activity Component

Huge Thanks to the BuddyPress core developpers !

 apeatling

 johnjamesjacoby

 MrMaz

 DJPaul

 boonebgorges

BuddyPress is definitely my favorite WordPress plugin. "Social Networking in a box" and this promise is at the « rendez-vous ».

From my point of view, one of the most important date of this plugin history is on 2010/02/16, when was launched the version 1.2. Thanks to it we no longer need a multi-site WordPress config to offer our visitors a community website.

BuddyPress is a very singular plugin : to such an extent that we talk of "BuddyPress Developers" ! I don't know a lot of plugins for which such qualification is used to define persons who engage in the design of extensions to this plugin.

And precisely what makes BuddyPress 'not like the others' is the multitude of hooks (like Action or Filter) that are available and allows us to extend it by our creations.

Since I met it in July 2010, I've written 25 articles to submit tips or plugins of my design such as BP My Home, BP Show Friends, Bowe Codes or BP Code Snippets.

Today, my goal is not necessarily to deliver a new trick or plugin (even if there is one to download !). It is rather to make you want to create yours and join the hall of fame of BuddyPress plugin developers.

I invite you to follow me for a serie of articles about an expedition in the heart of BuddyPress. This first article will allow us to meet the landscape and design a plugin on what for me is the **Major** BuddyPress component : the Activities.

Table of Contents

Preparing our surviving kit!

Designing a BuddyPress plugin requires first to be equipped. Here's my toolbox:

1. WordPress 3.3.2 and BuddyPress 1.5.5.
2. A local Apache / MySql / Php server (I chose MAMP).
3. A text editor (i like TextMate)
4. bookmarks to BuddyPress Codex and especially to the conditional template tags section.
5. A bookmark to WordPress Codex of course.
6. PoEdit for the internationalization of our plugin
7. Not to mention a good playlist. Right now I listen to RHCP

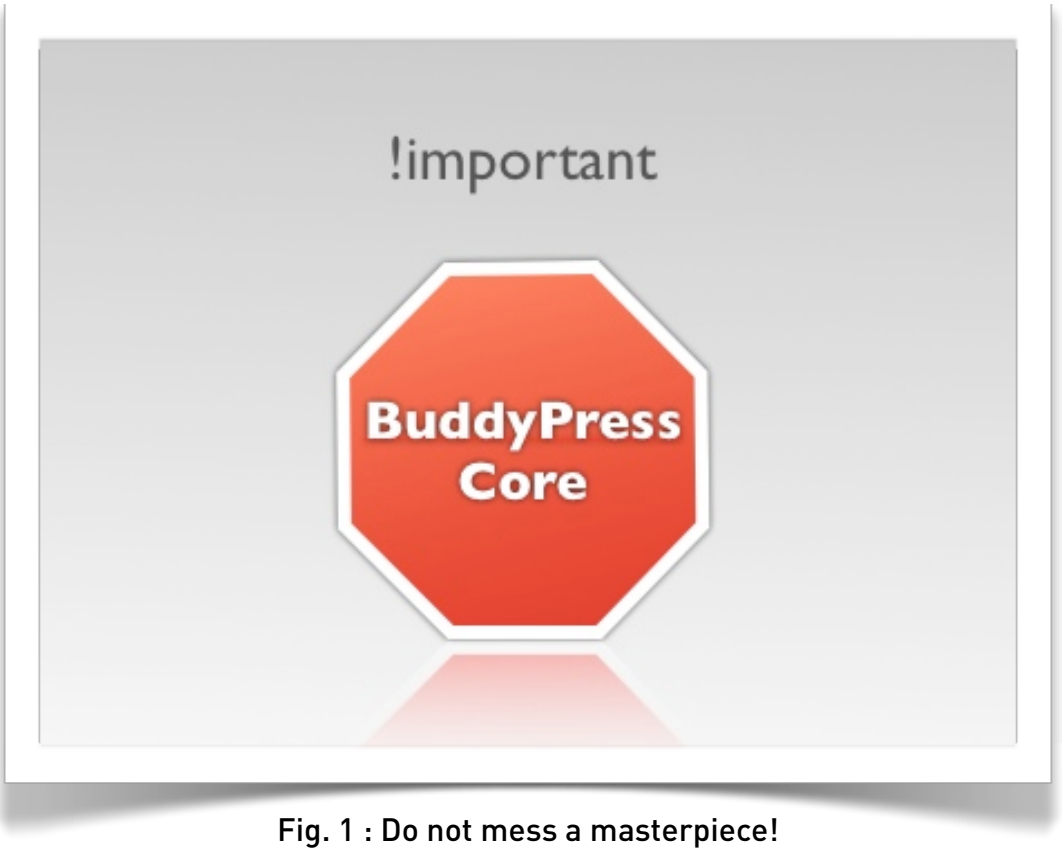


Fig. 1 : Do not mess a masterpiece!

It's essential to avoid (**i'd rather say it's forbidden**) modifying the source code of BuddyPress. Beyond the risks to its stability, consider upgrading! BuddyPress is regularly publishing upgrades and if you modify it, you'll lose all your changes at its next upgrade.

Landing! Enjoy the lanscape.

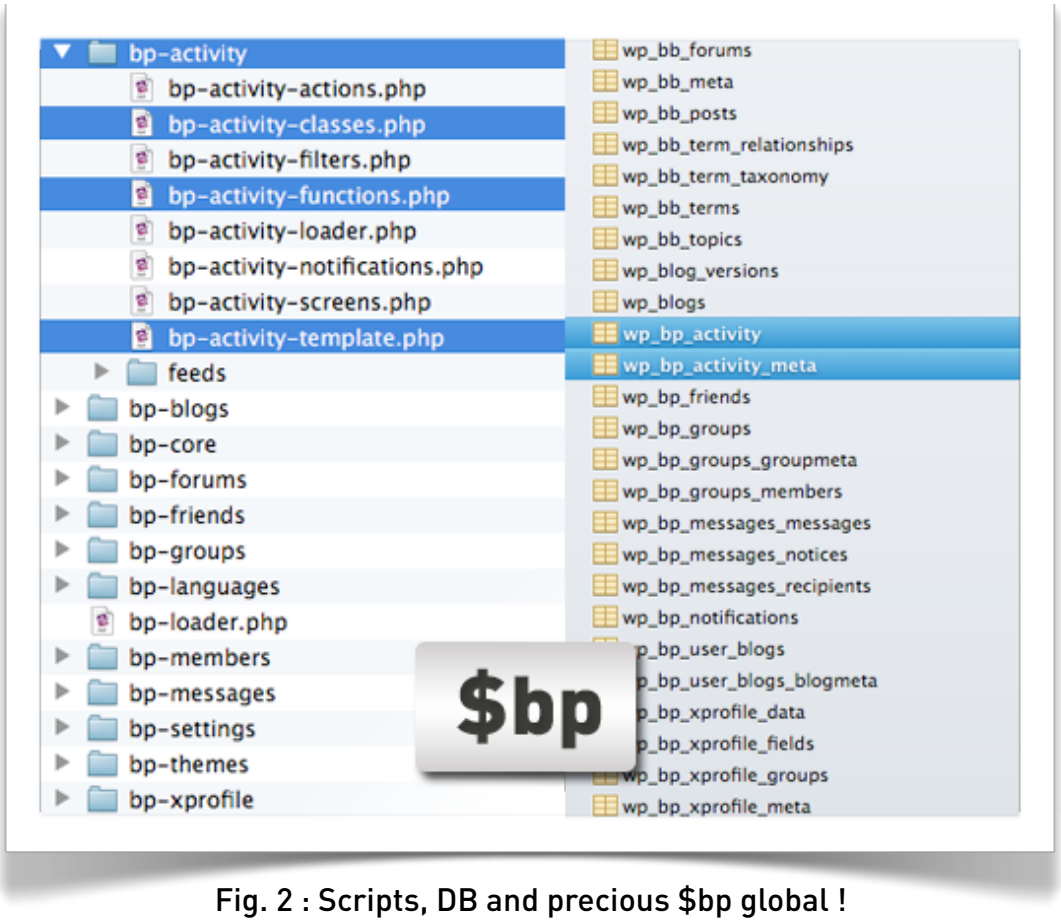


Fig. 2 : Scripts, DB and precious $bp global !

BuddyPress is an extension that runs across the network when WordPress Mu config is set, and its views are available on main blog's network. Note that since version 1.5, BP creates WordPress pages for each of its component.

Highlighted in blue, the tables and scripts that will be useful for our plugin. Due to lack of space, I have not detailed the content of the bp-themes directory which contains the default theme for BuddyPress.

We will focus in particular on these 2 activity templates (entry.php and post-form.php).

We can count on the precious **$bp** global which stores a bunch of useful variables that will help us in our plugin. To use it, simply refer to it at the beginning of your function like this:

```
function myfunction() {
     global $bp;
     / * your code * /
}
```

Here are three variables of **$bp**, I use regularly:

- `$bp->groups->current_group->id`: informs about the id of the displayed group
- `$bp->displayed_user->id`: informs about the id of the displayed member
- `$bp->loggedin_user->id`: informs about the id of the connected member

We can also extend BuddyPress by using its **BP_Component** class to design our own components, the **Group API** to add simple "apps" for groups and a number of very interesting functions to interact with all its components (activities, notifications, groups, forums, profiles ...). I will soon talk about it in the next episode of this serie.

Building the first version of our plugin

Our goal: to enable the super admin to use a 'support' group in order to share support messages on all the other groups of the BuddyPress driven community website.

demo available here http://vimeo.com/imath/bp-agu-0-1

Let's start by building our test environment: create 3 groups, 2 public and 1 private. Our "support" group will be public, we will first store its ID as a constant in our plugin.

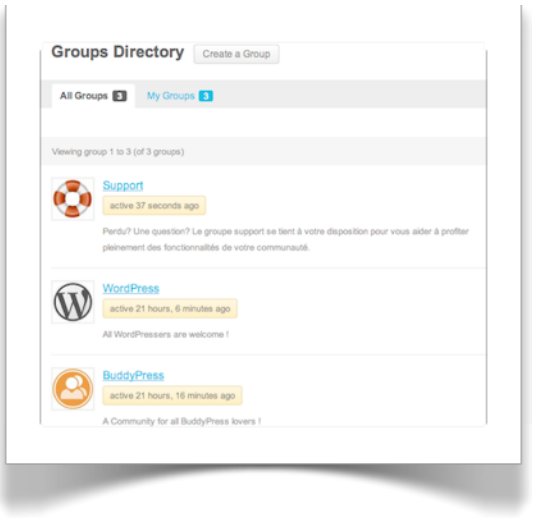
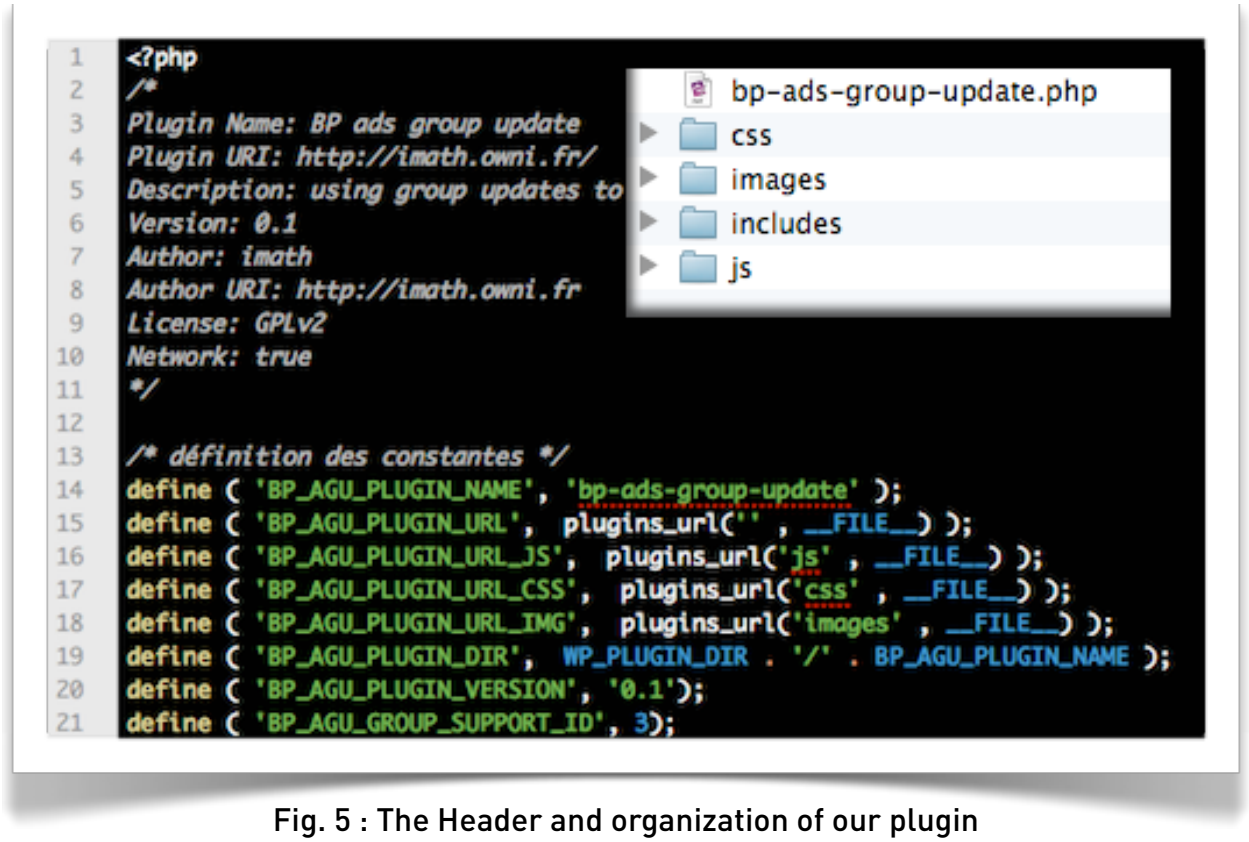


Fig. 4 : The 3 groups you'll need to create to test our plugin (**Support group must be public**)

In the WordPress plugins directory, we create a new folder called **bp-ads-group-update** in which we will add the 4 sub-folders : **js**, css, images, and includes. The main file of our plugin is named bp-ads-group-update.php. Let's first edit this file so that WordPress understands it is a plugin by adding specific tags to its header.




```php
<?php
/*
Plugin Name: BP ads group update
Plugin URI: http://imath.owni.fr/
Description: using group updates to
Version: 0.1
Author: imath
Author URI: http://imath.owni.fr
License: GPLv2
Network: true
*/

/* définition des constantes */
define ( 'BP_AGU_PLUGIN_NAME', 'bp-ads-group-update' );
define ( 'BP_AGU_PLUGIN_URL',  plugins_url('' , __FILE__) );
define ( 'BP_AGU_PLUGIN_URL_JS',  plugins_url('js' , __FILE__) );
define ( 'BP_AGU_PLUGIN_URL_CSS',  plugins_url('css' , __FILE__) );
define ( 'BP_AGU_PLUGIN_URL_IMG',  plugins_url('images' , __FILE__) );
define ( 'BP_AGU_PLUGIN_DIR',  WP_PLUGIN_DIR . '/' . BP_AGU_PLUGIN_NAME );
define ( 'BP_AGU_PLUGIN_VERSION', '0.1');
define ( 'BP_AGU_GROUP_SUPPORT_ID', 3);
```

Fig. 5 : The Header and organization of our plugin

Just like BuddyPress, our plugin will run throughout the Network if WordPress is configured in multi-site mode (Network tag is set to true) . This will prevent its display in the list of available extensions of child blogs, if the super admin has allowed plugins menu in the network options. Our BP_AGU_GROUP_SUPPORT_ID constant will store the support group ID (in this example 3). I also added other constants to more easily refer to js, css and images urls. The plugins_url() function is very convenient because according to the protocol of your website (http or https), it will automatically build the right url to the targeted files.

Warm-up!

In the WP administration (or Network administration) plugins menu activate the plugin. Of course, at this moment nothing happens, but we can start having some fun. If you want to know the list of variables of the **$bp** global, you can hook **wp_head** in order to print this list.

```
add_action('wp_head', 'bp_agu_dump_bp');

function bp_agu_dump_bp() {
     global $bp;
     ?>
     <pre> <?php var_dump($bp);?></pre>
     <?php
}
```

If you refresh the front of your blog, the result will show the ultimate design experience ever !
More seriously, the very beginning of the code shows an example of intercepting an action hook. Using the function **add_action('hook_to_intercept', 'by_my_fonction', 9, 1)**, we tell WordPress as soon as it passes through the '**hook_to_intercept**' then it must run '**by_my_fonction**' code. The two following arguments are used to indicate the execution priority of our function and the number of arguments to get from '**hook_to_intercept**'.
You can delete this function as we will not use it in our plugin..

Load a javascript only when needed

In our toolbox, let's use our "conditional template tags" bookmark. Whenever it's possible, for your plugin, I invite you to always make sure to load your javascripts and style sheets only when you need it. As our support messages will be written from the activity/post-form.php template of our support group, which is by default included in its home page, we will use the conditional template tag **bp_is_group_home()**.

```
add_action('bp_screens', 'bp_agu_load_js');

function bp_agu_load_js() {
     global $ bp;
     if (bp_is_group_home() && $bp->groups->current_group->id ==
BP_AGU_GROUP_SUPPORT_ID) {
          wp_enqueue_script('bp-agu-js', BP_AGU_PLUGIN_URL_JS.
'/ bp-agu.js', array ('jquery'));
     }
}
```

Here is a very useful new WordPress feature: wp_enqueue_script('**id_of_our_js**', '**url_to_my_js**', array('**dependencies**')).
WordPress recommends using this function to include our scripts, one advantage is that it allows us to include an identified script once and easily manage its dependencies (in our example jQuery).

For information, to include a style, you can use wp_enqueue_style.
To load this javascript we must choose the right time and ensure that all variables that we will be tested have already been initialized. Here I use the 'bp_screens' hook. It is located in the bp-core/bp-core-hooks.php script. This is the hook used to define templates and display BP components page. If, for example, I choose the **plugins_loaded** hook, nothing will happen because the variable **$bp->groups->current_group->id** is not initialized yet.

Actually, since we havn't created the javascript file, nothing will happen! So let's create the file bp-agu.js in our js directory. If you want to test, simply edit its content this way:

```
jQuery(document).ready(function($) {alert ('loaded!')} );
```

Finding the best hook to intercept

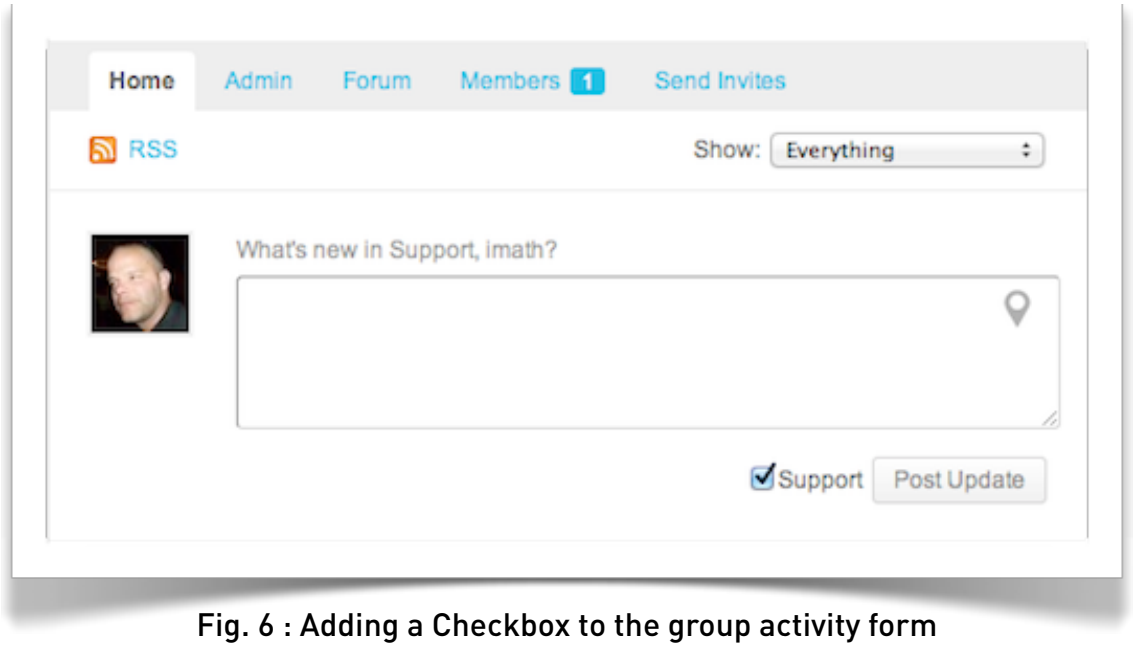


Fig. 6 : Adding a Checkbox to the group activity form

We'll come back in a few lines to our js file. For now, we can easily add a checkbox to the support group activity form. I invite you to browse the template activity/post-form.php in the bp-default theme folder of BuddyPress.

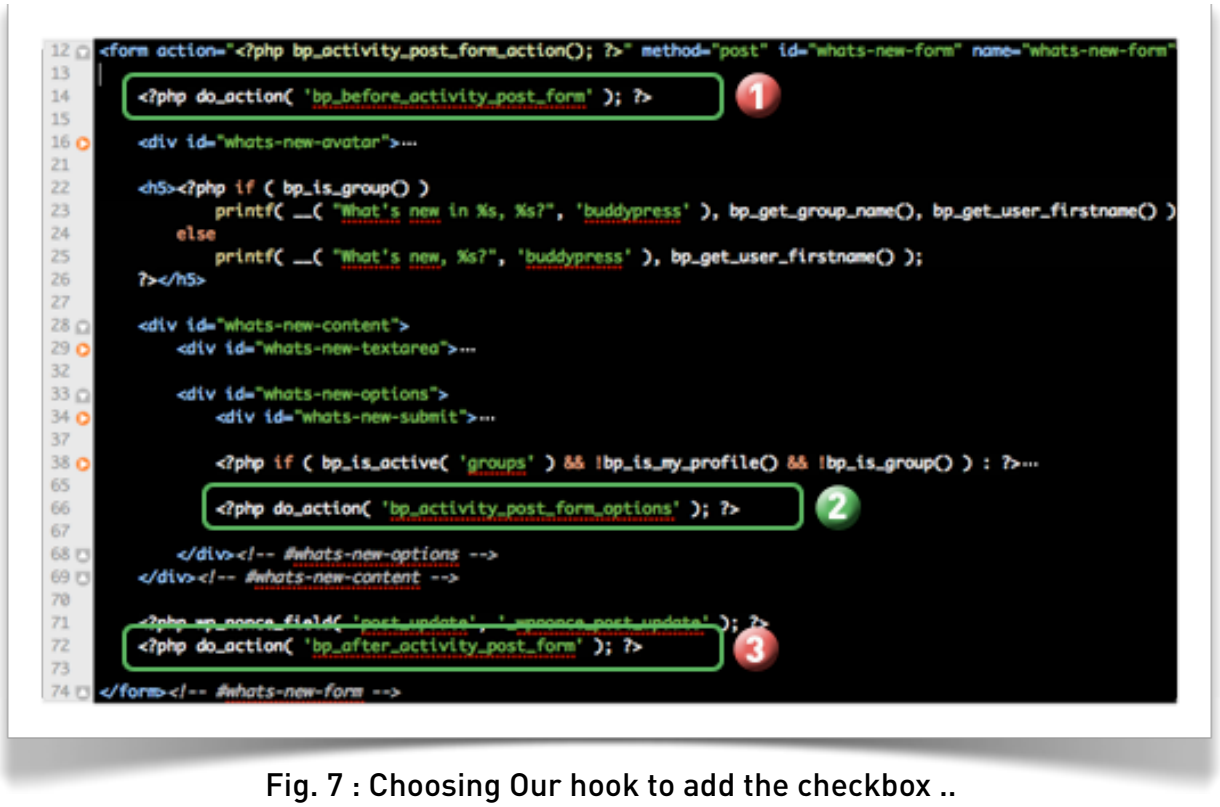


Fig. 7 : Choosing Our hook to add the checkbox ..

The above picture shows three hooks. If you test number 1 and 3, you'll find that the positioning of our checkbox is either too high or really too low. So we will use the '**bp_activity_post_form_options**' hook . Our function to add the checkbox will be:

```
add_action('bp_activity_post_form_options', 'bp_agu_checkbox');

function bp_agu_checkbox() {
     global $bp;
     if(is_super_admin() && bp_is_group_home() && $bp->groups->current_group->id == BP_AGU_GROUP_SUPPORT_ID) {
     ?>
     <span id="bp-agu-cb"><input type="checkbox" value="1" name="_support_message" id="support_message">Support</input></span>
     <?php
     }
}
```

The is_super_admin() function returns **true** if the logged in user is the Admin or Super Admin if WordPress is configured in multi-site mode. If you test this on the front side, you'll find that your checkbox is not right next to the submit button as shown on Fig 6. This is normal, since we will enrich this method using jQuery in the first version of our plugin.

Our checkbox (if checked) will serve to trigger the behavior of sharing the support message on all the BuddyPress groups of our website. We never know, **all updates of this group may not be support messages**.
The activity recording related to the publication of new forums would otherwise be automatically shared on all groups, and we do not want this behavior. Let's now intercept the activity, once published, to test the value of this checkbox.

The function responsible of activity recordings is **bp_activity_add ()**. It is located in the BuddyPress script bp-activity/bp-activity-functions.php. At the end of this function, a hook allows us to intercept activity datas as soons as it's published, it is the **do_action ('bp_activity_add')** marker. However, if we hook activity publishing here, we will have to test if the activity is a group activity or not. There's a simpler way. In the bp-groups/bp-groups-functions.php script, the **do_action ('bp_groups_posted_update')** hook of **groups_post_update()** function allows us to be sure the activity is a group activity.

```
/ * Function for temporary testing purposes .. * /

add_action('bp_groups_posted_update', 'bp_agu_check_support', 9,
4);

function bp_agu_check_support($content, $user_id, $group_id,
$activity_id) {
     if ($group_id == BP_AGU_GROUP_SUPPORT_ID &&
$_REQUEST['_support_message'] ) {
     wp_die('the value of the checkbox is'.
$_REQUEST['_support_message']);
     }
}
```

In the above code, in red, I put a 4th argument to my **add_action()** to indicate that I want to get the four arguments that are available in the function **do_action('bp_groups_posted_update')**. In my **bp_agu_check_support** function, i list these four arguments so that i'll be able to manipulate them.

I invite you to test this function in two steps: first normally, then by disabling javascript in your browser. In the first case, nothing happens, in the second case, a beautiful WordPress error will display the value of our checkbox. BuddyPress regularly uses AJAX requests to dynamize the user experience, so we'll have to use an alternate method to get the value of our checkbox, and still, we'll have to handle the case when javascript is disabled.

What about a small cookie break ?

To send some of its variables in AJAX, BuddyPress uses cookies that lasts the duration of your session. If you open the inspector of your browser when you have filtered the activities to only show updates you will see a cookie named **bp-activity-filter** whose value is fixed at **activity_update**. That's why when you go in one of your groups or in your profile the activity are also filtered on updates only.

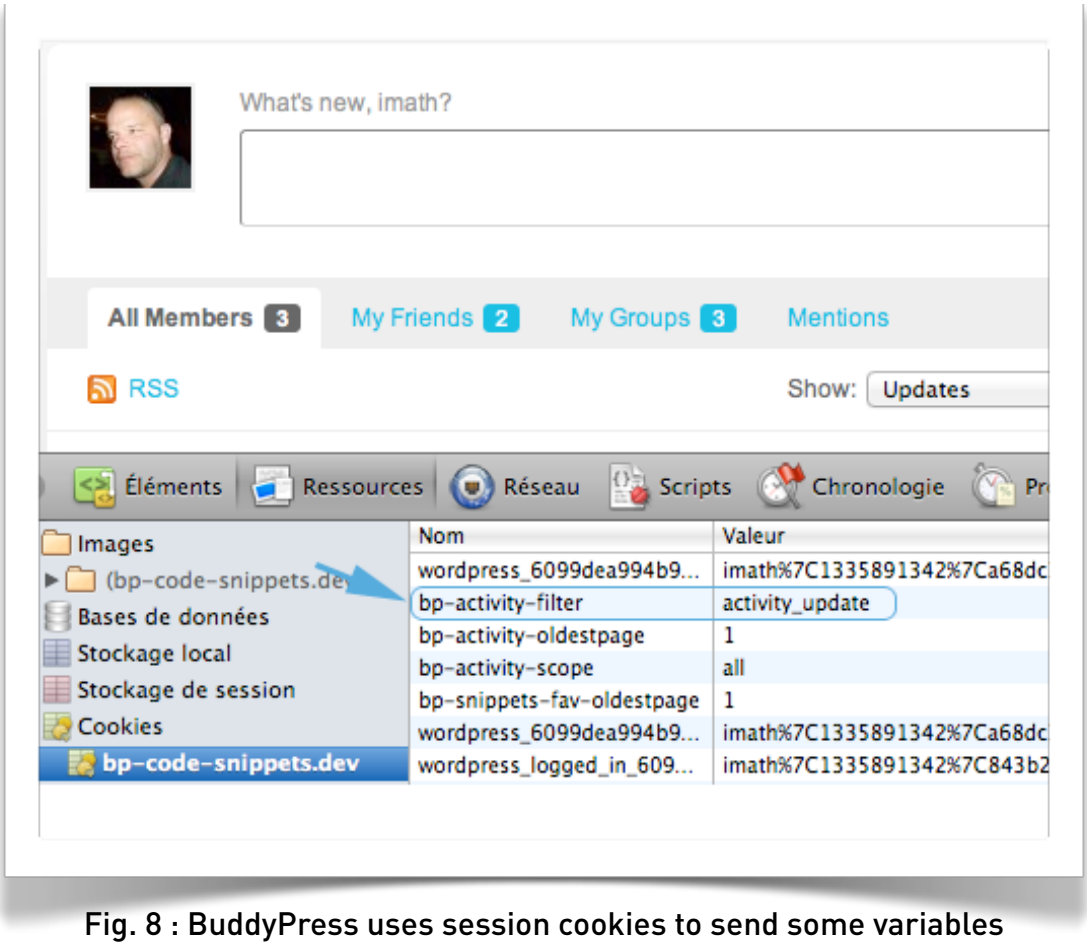


Fig. 8 : BuddyPress uses session cookies to send some variables

We will therefore use this mechanism to pass the value of our checkbox in the AJAX function that handles post activities. Watch the extract of the BP default theme _inc / global.js file, and especially line 67:

Fig. 9 : $_POST ['cookie'] will save us

Awesome! When an activity is posted in AJAX, BuddyPress encode all session cookies and send this data in the variable **$ _POST ['cookie']**. We just need to write a cookie that we'll call **bp-agu-is-support** as soon as the checkbox is checked. And to do that, we will edit our javascript file that was only alerting "loaded!" so far. And as we're using a javascript, we will take benefit of it to move our checkbox right next to the submit button. Here is our brand new bp-agu.js:

```
jQuery(document).ready(function($) {
     $('#whats-new-submit').prepend($('#bp-agu-cb').html());
     $('#bp-agu-cb').html("");

     if ($.cookie("bp-agu-is-support") == 1) {
          $("#support_message").attr('checked', 'true');
     }

     $('#support_message').live('click', function() {
          if ($(this).attr('checked'))
               $.cookie("bp-agu-is-support", $(this).val());
          else
               $.cookie("bp-agu-is-support",'');
     });
});
```

In bold blue, the moment when we write the value of our checkbox in our cookie. Now, we need to test the value of this cookie in order to inform the Admin if he's about to post a support message in all groups or simply an update in its support group. That's why, we first test if the cookie is set and activate the checkbox eventually. Now that the javascript is in place, we can take care of our **bp_agu_check_support** function to make it do something smarter than throwing a WordPress error!

On our way to the "main street" of our extension

```php
function bp_agu_check_support( $content, $user_id, $group_id, $activity_id ) {
    global $wpdb;

    $is_support_message = $_REQUEST['_support_message'];

1   $_BP_COOKIE = wp_parse_args( str_replace( '; ', '&', urldecode( $_POST['cookie'] ) ) );

    if( $_BP_COOKIE['bp-agu-is-support'] == 1)
        $is_support_message = $_BP_COOKIE['bp-agu-is-support'];

    if( $group_id == BP_AGU_GROUP_SUPPORT_ID && $is_support_message ) {

2       $support_activity = bp_activity_get_specific('activity_ids='.$activity_id);

        $support_args = array (
            'user_id'           => $user_id,
            'action'            => $support_activity['activities'][0]->action,
            'content'           => $support_activity['activities'][0]->content,
            'primary_link'      => $support_activity['activities'][0]->primary_link,
            'component'         => 'groups',
            'type'              => 'support_update',
            'item_id'           => false,
            'secondary_item_id' => $activity_id,
            'recorded_time'     => $support_activity['activities'][0]->date_recorded,
            'hide_sitewide'     => 1 /* this way on site wide activities only the support g
        );

        /* gets groups ids and loops to duplicate activity on each of them (except the suppo

        $group_ids = $wpdb->get_col("SELECT id FROM {$wpdb->base_prefix}bp_groups");

3       if( count( $group_ids ) >= 1) {
            foreach( $group_ids as $grp_id ){

                if($grp_id != $group_id) {
                    $support_args['item_id']=$grp_id;

                    groups_record_activity( $support_args );
                }
            }
        }
    }
}
```

Fig. 10 : here is our new bp_agu_check_support function

To get the code of this function and eventually copy and paste it in the plugin we started, you can always download the sources . Now analyze the three steps of this function.

First, we retrieve the value of our checkbox (step 1). We begin by assuming that the user has disabled javascript in his browser so we initialize the variable `$is_support_message` with the post value of the checkbox. If it does not exist then, javascript is enabled. So we'll need to parse `$ _POST['cookie']` to store all the cookies of the document in an associative array. Then, we get the value of our cookie **bp-agu-is-support** and assign it to our variable `$is_support_message`.

Step 2: if our cookie is set and if the group ID - which has been passed as parameters to our function via the arguments of the marker do_action ('bp_groups_posted_update') - corresponds to our support group ID (defined as a constant at the beginning of our plugin), then we use **bp_activity_get_specific()** function - which is located in the BuddyPress bp-activity/bp-activity-functions.php script - to get the elements of the activity.
As its name suggests, it retrieves activities by passing either an id or an array of activities id. In the same way that we inherited from the hook we're intercepting the group_id, the activity ID is in the **$activity_id** argument of our function. We now need to prepare the array of arguments the **groups_record_activity()** function expects. You'll find these function in the bp-groups/bp-groups-activity.php script. We will simply forget to specify the index '**item_id**' of this array for now. Important, please note that at line 75 we've created a new type of activity: **support_update**

Step 3. Reminder : our goal is to enable the super admin to use a 'support' group in order to share support messages on all the other groups of the BuddyPress driven community website.
To achieve this, a loop is needed. This is what we're about to do, thanks to a very useful WordPress class that eases communication with the database: $wpdb . By the way, you've noticed that I referenced it at the beginning of the code (at line 56) of this function? For a quick brief on this class, I invite you to read from slide 11 to 15 of the presentation I used during the WordPress Algeria day. So we get an array containing the ids of all groups (Table **wp_bp_groups**), on which we loop to add the famous index '**item_id**' - having checked that the group id is not the one of our support group - of the array of arguments expected by the **groups_record_activity()** function and while still in the loop we run that function. As a result the activity is duplicated on our two groups.

Being imaginative using tricks

Then you would say "yep, but if I go on the Site Wide Activities, i'll have three activities displayed instead of one : not great!".
Except that in the argument array of **groups_record_activity()** function, we specified that these activities would be hidden (**'hide_sitewide' => 1**).

Fig. 12 : Oops, we still have some work !

You couldn't wait to test it, neither could i ! First I checked the Site Wide Activities of BuddyPress: and only one message support is displayed, which is normal because we specified (`'hide_sitewide' => 1`). If I go to the private group that we created for our tests, the support acivity is also displayed. But, when I go to the public group of our test environment: that's an epic fail!

Why so much hate? It's been 10 minutes you're reading this tutorial and "dang, not working at all"! In fact it is normal the public group does not display this activity because in the table **wp_bp_activity,** the hide_sitewide field of this activity is set to **1** and as we're in a public group, it will display only the updates with hide_sitewide field set to **0**.

So, we need to go a little deeper into the heart of the Activity component of BuddyPress. The **groups_record_activity()** function located in bp-groups/bp-groups-activity.php script calls bp_activity_add() function from bp-activity/bp-activity-functions.php script which calls the BP_Activity_Activity class located in bp-activity/bp-activity-classes.php script to create the activity. Still following me?

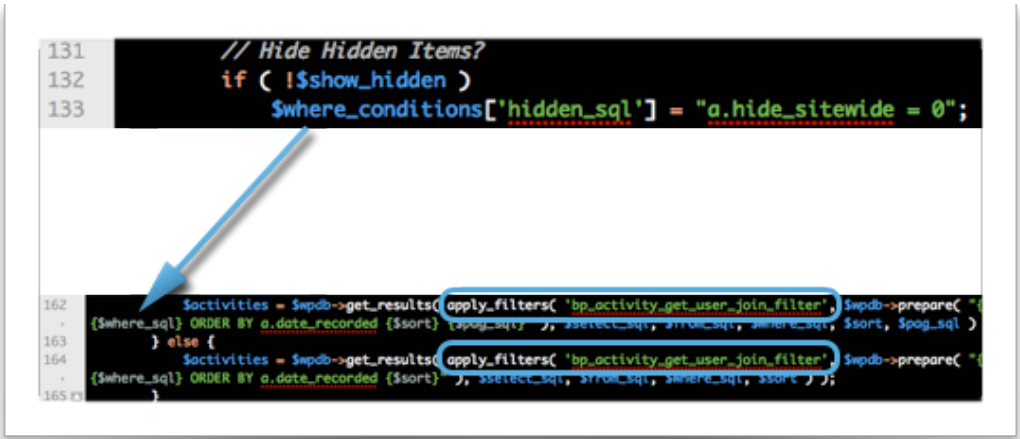


Fig. 13 : Extract from the get() function of BP_Activity_Activity class

The **get()** function of this class creates this Sql WHERE argument "**a.hide_sitewide = 0**". But if the group is not private: we actually need to have the updates with a hide_sitewide field that is greater than or equal to 0!
I suggest a trick to achieve this. You noticed that the query that populates **$activities** has a filter hook? So we will be able to use it. The characteristic of this type of hook is that it expects a value in return. So we will use a add_filter() to get the value of the request and return to it a slightly different query: admire the trick!

```
add_filter('bp_activity_get_user_join_filter',
'bp_agu_sql_trick_to_include_hidden', 99, 1);

/ * Do not forget the activities sql count! * /
add_filter ('bp_activity_total_activities_sql',
'bp_agu_sql_trick_to_include_hidden', 99, 1);

function bp_agu_sql_trick_to_include_hidden($sql) {
     if(bp_is_group_home()) {
          return str_replace('a.hide_sitewide = 0',
'a.hide_sitewide >= 0', $sql);
     }

     return $sql;
}
```

If we are on the group home, we will replace **a.hide_sitewide = 0** by **a.hide_sitewide >= 0**, otherwise it returns the query without modification. As for the add_action() hook, we're specifying the number of arguments expected (1) after the priority, so that the value of the SQL query is passed to our **bp_agu_sql_trick_to_include_hidden**($sql) function. That will satisfy our needs.

Loopings!

Fig. 14 : Ouch, Caramba!

This is embarrassing .. All is not quite finished: if from the Site Wide Activities of our site, I activate the My Groups tab or if I go on my profile activities, being super Admin, three identic activities are displayed. So we could say "it's okay, it only affects the super Admin!", But the house tries to heal its plugins. So we will solve this new problem.
To do so, let's meet the BuddyPress Activity loop. Let's look at the bp-default theme template activity/activity-loop.php.

Fig. 15 : Well here is a very nice loop

bp_has_activities() function will gently bring us to the bp-activity/bp-activity-template.php script which contains BP_Activity_Template class.
bp_has_activities() calls this class and stores its rendering in the global **$activities_template**. This global allows BuddyPress to return various informations related to the activity whenever we will use the activity template tags. If you read the template activity/bp-entry.php of the default theme, you will meet a lot of these template tags, here's a selection that will be useful for our plugin:

- bp_activity_can_comment(),
- bp_activity_can_favorite(),
- bp_activity_user_can_delete(),
- **bp_get_activity_type(),**
- bp_activity_user_link(),
- bp_activity_avatar(),
- **etc. ..**

Interesting detail about these template tags : it often works in pairs. Let me explain: bp_activity_avatar() displays the result of bp_get_activity_avatar(). And all bp_get_ ... contain filters to hook on.

Back on **bp_has_activities()**. In lines 332 to 371 of its code, we see that depending on the "scope", BuddyPress prepares adjustments to the query that we have seen above. Thus, in the case of activities displayed on the logged in user's profile, all activities he shared will be integrated, it is the same for the activities of the My Groups tab in the Site Wide Activities.
And the function that directs this switch is both in the blue box of figure 15 and both (mainly actually!) in bp-core/bp-core-template.php script, i'm talking of **bp_ajax_querystring**, and guess what, this function provides a filter : we will be able to change this behavior from our plugin.

```
add_filter('bp_ajax_querystring',
'bp_agu_neutralize_support_updates_in_scopes', 99, 2);

function bp_agu_neutralize_support_updates_in_scopes
($ajax_querystring, $object) {

     $r = wp_parse_args($ajax_querystring);
     extract($r);

     if ($scope == 'groups' || bp_is_user_activity()) {
          $exclude_ids = array();
          $exclude_activities = bp_activity_get(array
('show_hidden' => true, 'filter' => array ('action'
=>'support_update')));

          foreach ($exclude_activities['activities'] as
$exclude_id) {
               $exclude_ids[] = $exclude_id->id;
          }

          return $ajax_querystring. '&exclude='. implode(',',
$exclude_ids);

     }

     return $ajax_querystring;
}
```

So in order to display only one activity, we simply exclude them if they have a **support_update** type. We did well when we created this type of update in our bp_agu_check_support() function ! So we put the activities to exclude into an array and add a variable to the AJAX query string that will contain the list of activity ids separated by commas.

A little makeover is needed ..

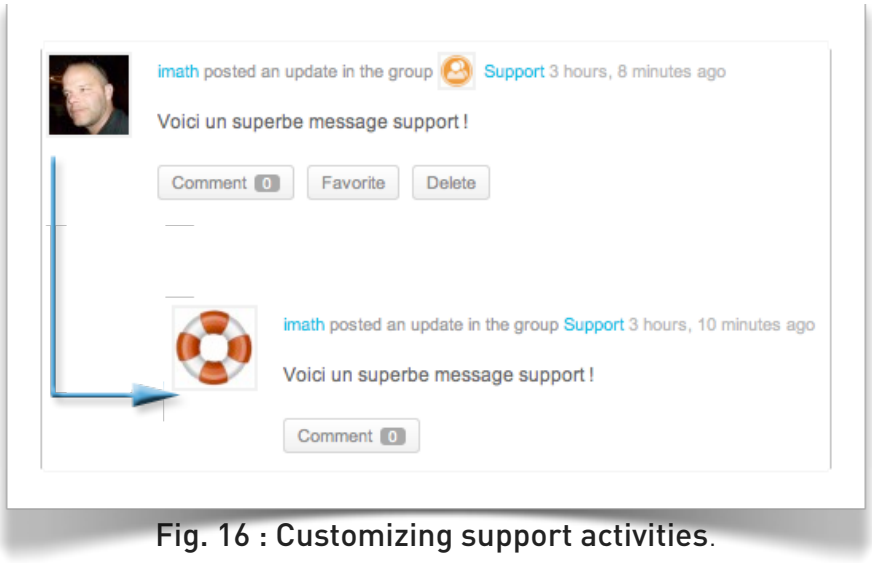


Fig. 16 : Customizing support activities.

Amazing what can be done with filters, right? Here the extent of the makeover for our support_updates activities that wil be displayed in groups:

1. Hiding traditional action buttons (comment, favorite or delete).
2. We change the avatar of the user by the support group one (and the userlink by the group home one).
3. Let's remove the secondary avatar since we do not need it anymore.
4. We modify the permalink of the support update for the support group's parent activity.
5. We add a comment button that redirects on the parent activity (the one created by the support group), and displays its number of comments.

All these mods by simply using filters of the activity template tags we discussed earlier. I will not illustrate each of them, but you will find all the customizations in the main file of the plugin available in download , here's an example:

```
add_filter('bp_activity_permalink',
'bp_agu_override_activity_time_since', 99, 1);

function bp_agu_override_activity_time_since($content) {
     if ('support_update' != bp_get_activity_type())
          return $content;

     $parent_activity_id = bp_get_activity_secondary_item_id();
```

```
     return preg_replace("/\/p\/([0-9]+)\//','/p/'.
$parent_activity_id.'/', $content);

}
```

Once the makeover is completed, members will not comment on **'support_update'** activities but will be redirected to the parent activity the support group created: this way we can centralize all the comments in one place.
By disabling the delete button we avoid its removal by group administrators.
Finally, we remove the favorite button because who wants to bookmark a support message?!

We still have to consider the parent activity deleting. If the super Admin deletes the support message all the logic falls down: you lose the mother! The following function takes care of this possibility by removing the child activities if the mother was to disappear ... Sad fate for these children, they are doomed not to outlive their parents..

```
add_action('bp_activity_delete', 'bp_agu_handle_deleting', 9,
1 );

function bp_agu_handle_deleting($args) {
     bp_activity_delete ( array('type' => 'support_update',
'secondary_item_id' => $args['id'] ) );
}
```

Let's ease the the super admin's life and take care of deleting our fingerprints

That's not bad, it can almost be our first package for testing! Before, there must be an admin interface for our plugin in the WordPress backend, so that Admin does not have to modify the source code of the plugin to adapt the constant **BP_AGU_GROUP_SUPPORT_ID** to match the id of his support group (which is not necessarily 3!).
As we're designing a BuddyPress plugin, I invite you to link this interface to the BuddyPress administration. Depending on the configuration of WordPress, this menu is either in the WP Admin area or in the Network Admin one. To anticipate this eventuality, we can use the **is_multisite()** function to intercept the good hook depending on its result.

```
add_action( is_multisite() ? 'network_admin_menu' :
'admin_menu', 'bp_agu_backend_menu', 21);

function bp_agu_backend_menu(){
     if ( !is_super_admin() )
          return false;
```

```
    add_submenu_page ( 'bp-general-settings', 'Options Ads
Group Update', 'Options Ads Group Update' , 'manage_options',
'bp-agu-backend-slug', 'bp_agu_backend_page' );

}
```

For more information about adding Administration page in WordPress, I invite you to visit(again) the presentation I used during the WordPress Algeria Day and more precisely its slide 36. In red in the above code, the reference to the function that plays when the administrator clicks on the "Options Group Update Ads" menu.

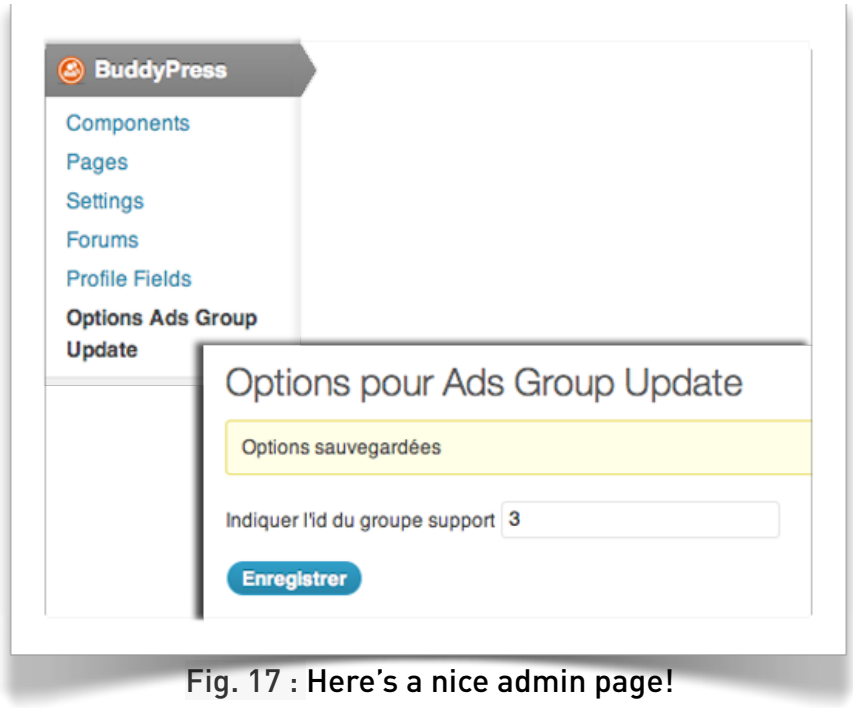


Fig. 17 : Here's a nice admin page!

If we look at the source code of the **bp_agu_backend_page()** function, you will see that we use the security mechanism wp_nonce to ensure that the request comes from our site. It is also important to validate user inputs before storing or displaying them. In our case, as we're expecting a group ID, or an integer, we use the function **intval()** to ensure that only an integer will be saved.

```
function bp_agu_backend_page() {

    if($_POST['_bp_agu_options'] && check_admin_referer('bp-
agu-option', '_bp_agu_option') ) {

    if( update_option('_bp_agu_group_support', intval
($_POST['_bp_agu_group_support']) ) !== false )
        echo '<div id="message" class="updated"><p>Options
saved</p></div>';
```

```
	}

	$group_support =
intval(get_option('_bp_agu_group_support') );

	/* to be continued... */

}
```

We use the wp_options table to store the global settings of our plugin. Now that the admin can store its group ID, do not forget to change the value of our constant **BP_AGU_GROUP_SUPPORT_ID**!

```
define ( 'BP_AGU_GROUP_SUPPORT_ID',
bp_agu_define_support_group() );

function bp_agu_define_support_group() {
	return intval ( get_option ('_bp_agu_group_support') );
}
```

We almost finished the beta version of our plugin: hang in there!
As the functionality of our plugin may evolve, in the WordPress wp_options table, it may be useful to store its version. This way, during updates, we can provide the appropriate actions to be performed depending on the version used by the site. We will use WordPress registration hook which occurs just after the activation of our extension by the administrator.

```
register_activation_hook ( __FILE__, ' bp_agu_activate ' );

function bp_agu_activate() {
	if( get_option ('_bp_agu_version') !=
BP_AGU_PLUGIN_VERSION ){

	update_option ( '_bp_agu_version', BP_AGU_PLUGIN_VERSION );

}
}
```

Finally and quickly: we will add a new file at the root of our plugin. We will call it uninstall.php. This file will be called by WordPress at the time the admin has confirmed its desire to remove the plugin from his site.
This is very important, as far as possible, to try to erase all traces of our plugin in particular in the database. Also in our case, we set up a mechanism that duplicates activities. As soon as the admin has deleted our plugin « patatra »...! all support_updates

we have cleverly hidden, will reappear. Below you'll find the download link of our plugin .. Hip Hip Hip ...

Download the first version of our plugin..

Half time !

Crédit Photo : Midsummer Nights's Cross Race: Winner de Hugger Industries, sur Flickr

If we check our landscape, we did meet **BP_Activity_Activity** "CRUD" Class of the **wp_bp_activity** table, functions for adding and deleting activities as well as the **BP_Activity_Template** templating class and the filterable activity template tags. There is a point we have not addressed: the activity metas that are stored in the **wp_bp_activity_meta** table. We will shortly be introduced to them in our next step.

Our plugin can still be improved: it is important to think about **internationalization**. This will increase its use and enrich the feedback and feature requests. It is also important to optimize it by ensuring that BuddyPress is "ready to rock" before loading the complete code for our extension.
By the way, this support updates functionnality made me immediately think of sponsored tweets, it's probably why i called this plugin Ads BP Group Update ..

Plugin Upgrade

Our new goal: would'nt it be nice if members could reshare activities just like we retweet..

demo available here http://vimeo.com/imath/bp-agu-1-0

Optimize!

Luckily, with this new reshare feature we can imagine the interest of adding a counter of number of reshared updates and then use the activity metas to store this counter. Before coding it, let's optimize and reorganize our plugin.

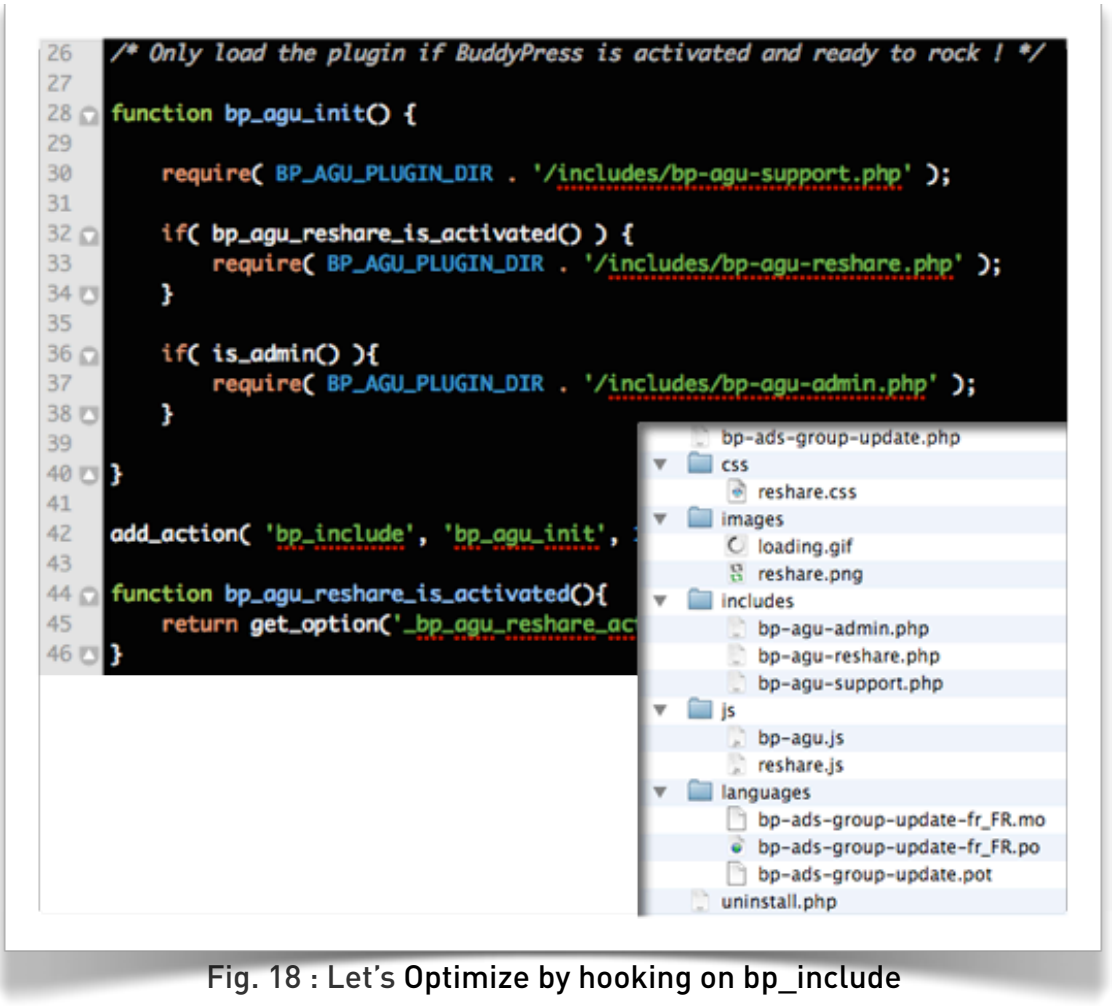


Fig. 18 : Let's **Optimize by hooking on bp_include**

As you can see, we've created three new files in the includes directory of our plugin. All functions related to the functionality **support_update** were moved in the includes/bp-agu-support.php script, we have prepared the includes/bp-agu-reshare.php script to

host the functions related to the **reshare_update** functionality and functions related to the administration interface of our plugin were moved in includes/bp-agu-admin.php.

bp-agu-init() is fired when BuddyPress passes through the **bp_include** hook, it first includes the file includes/bp-agu-support.php before loading if the conditions are right (reshare function enabled or admininterface) other scripts. By doing so, we can capitalize on the code of **support_update**. If you look at **bp_agu_reshare_is_activated function()**, it simply takes care of returning the choice we made in the administrative interface of our plugin. So our first step is to edit the code of the admin page by adding a radio to handle activation of **reshare_update**.

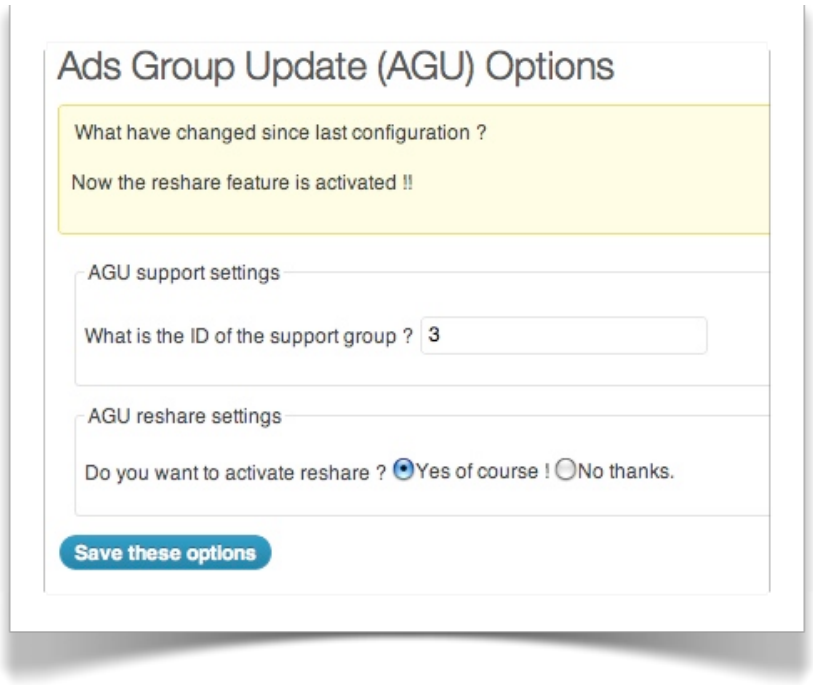


Fig. 19 : **reshare_update's on**

I will not detail here the code, you can consult it in the download below. We decide to better identify our "retweets" by using a new type of activity:'**reshare_update**'.

Capitalize!

To allow the member to reshare an activity, we simply add a new action button in the activity just like we did when we added a link to the comments of the parent activity (**support_update** cusomization item 5) Moreover, we will take advantage of this button that gets the number of comments of the parent activity and sends us back to its permalink. I think the comments would be much better in the first shared activity.

To do this in the includes/bp-agu-support.php script we'll need to give a higher priority on the comment button hook than for the reshare button hook we'll create. Moreover, we won't forget to add '**reshare_update**' type to the condition that creates this comment button.

```
/* in bp-agu-support.php */
// hook with a priority of 9
add_action('bp_activity_entry_meta', 'bp_agu_add_action_link',
9);

function bp_agu_add_action_link() {
     if( !in_array( bp_get_activity_type(),
array('support_update', 'reshare_update') ) )
          return false;

     /* to be continued */
}

/* in bp-agu-reshare.php */
// hook with a priority of 10, this button will be displayed
// after the comment one.
add_action('bp_activity_entry_meta',
'bp_agu_add_reshare_button', 10 );

function bp_agu_add_reshare_button() {
     global $bp;

     if( !is_user_logged_in() )
          return false;

     /**
      * we won't allow the reshare of support messages..
      * no interest..
      * /
     if('support_update' == bp_get_activity_type() )
          return false;
     /* to be continued */
}
```

Highlighted in yellow the condition "if not in the table" which we will also use to take advantage of points 1 and 4 of the support activities customization.

In our **bp_agu_add_reshare_button()** fonction, we will create a link containing a $_GET variable named 'to_reshare' to which we will add a nonce for security. This link will be

useful if the user has disabled javascript in his browser. By default, we'll use AJAX! Then, this button will be clickable if conditions are met:

1.The author of an activity won't be able to reshare it: no interest!
2. If the member has shared an activity, he won't be able to reshare it again: beside the "reshare" counter , we'll need a second meta activity that will store an array of user_ids who "retweeted" the activity

Fig. 20 : Reshare activity look and feel

WPAjaxifying the reshare

This is a very interesting WordPress mechanism. To use it, proceed in three steps :

1. Client-side (PHP -> JS): add a javascript variable to indicate the url that will handle ajax requests.
2. Client side (JS): provide a variable whose name is '**action**' in our jQuery.post.
3. Server side (PHP): Add a '**wp_ajax**' hook to intercept the variable '**action**' and print a response before stopping the PHP code execution.

In a BuddyPress environment, the variable "**ajaxurl**" is already available. So, next!
We take care of our javascript. Do not forget to load it into the page with **wp_euqueue_script** of course, do you remember how it's done?
This time, we not only need our javascript to be loaded if we're on the group activity page of a particular group but for all activities. The condition will be:

```
add_action('bp_actions', 'bp_agu_load_reshare_css_js ');
```

```
function bp_agu_load_reshare_css_js(){

     if( bp_is_activity_component() || bp_is_group_home() ) {
          wp_enqueue_style ('bp-agu-reshare-css',
BP_AGU_PLUGIN_URL_CSS.'/reshare.css');
          wp_enqueue_script ('bp-agu-reshare-js',
BP_AGU_PLUGIN_URL_JS.'/reshare.js', array('jquery'), 0, 1);
     }

}
```

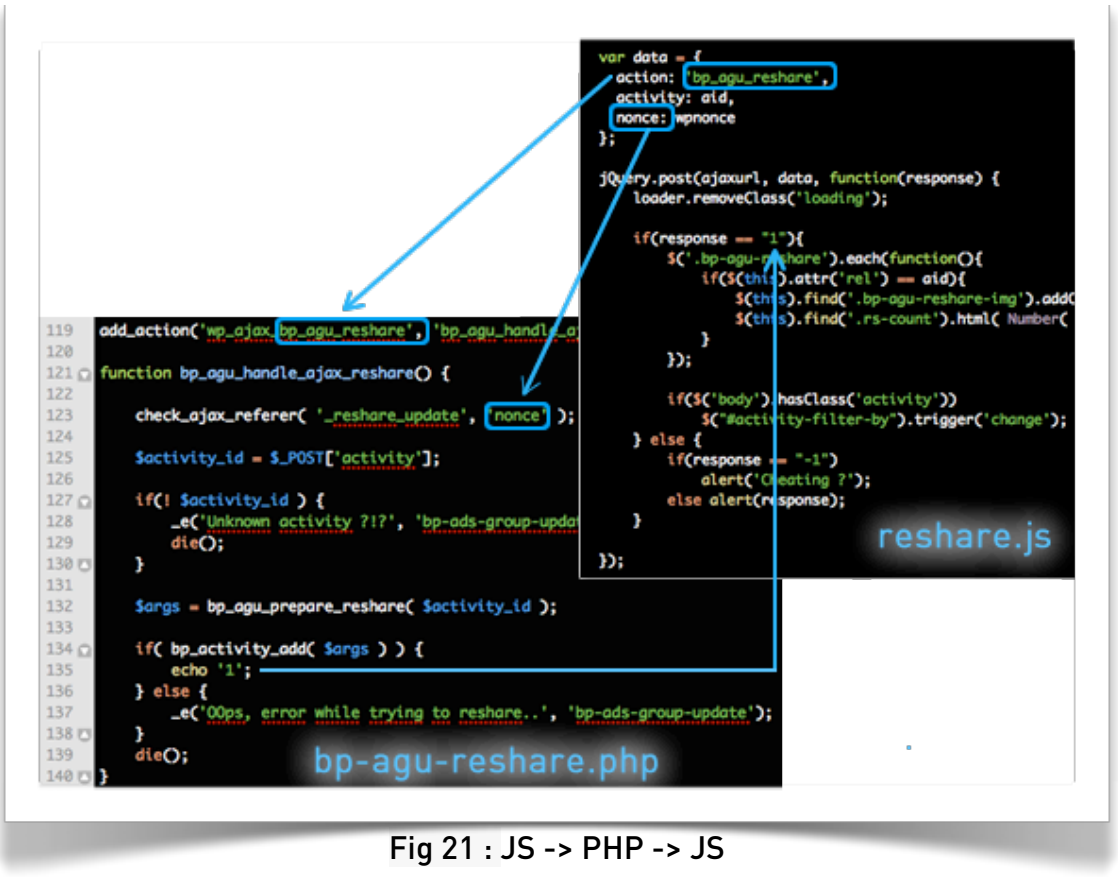


Fig 21 : JS -> PHP -> JS

reshare.js intercepts the click on the reshare button and sends the ajax request which is intercepted by **bp_agu_handle_ajax_reshare function()** localised in bp-agu-reshare.php script.

Once we've received the response, we change the button display and increment by 1 the number of "reshares". Finally, do not forget to return **false** to prevent the link to be actually submitted.

Enjoy the activity metas.

In the illustration above at line 132, **bp_agu_prepare_reshare()** function is called to build the array of arguments needed to record the '**reshare_update**' activity thanks to the BuddyPress **bp_activity_add()** function.
To do this it sends the id of the activity to reshare Ajax has given us. Now we'll be able to play with activity metas.

In the bp-activity/bp-activity-functions.php script, we find the three tools that allow us to interact with the wp_bp_activity_meta table.

- **bp_activity_update_meta**($activity_id, $meta_key, $meta_value)
  Adding or changing information stored in a meta key for the activity id.
- **bp_activity_get_meta**($activity_id = 0, $meta_key = "")
  It retrieves the information stored in the meta key for the activity id.
- **bp_activity_delete_meta**($activity_id, $meta_key = "", $meta_value = "")
  It deletes the information stored in the meta key for the activity id.

What is interesting is that the values are serialized / deserialized automatically using respectively **bp_activity_update_meta()** and **bp_activity_get_meta()**. This saves us time to store PHP arrays for example..

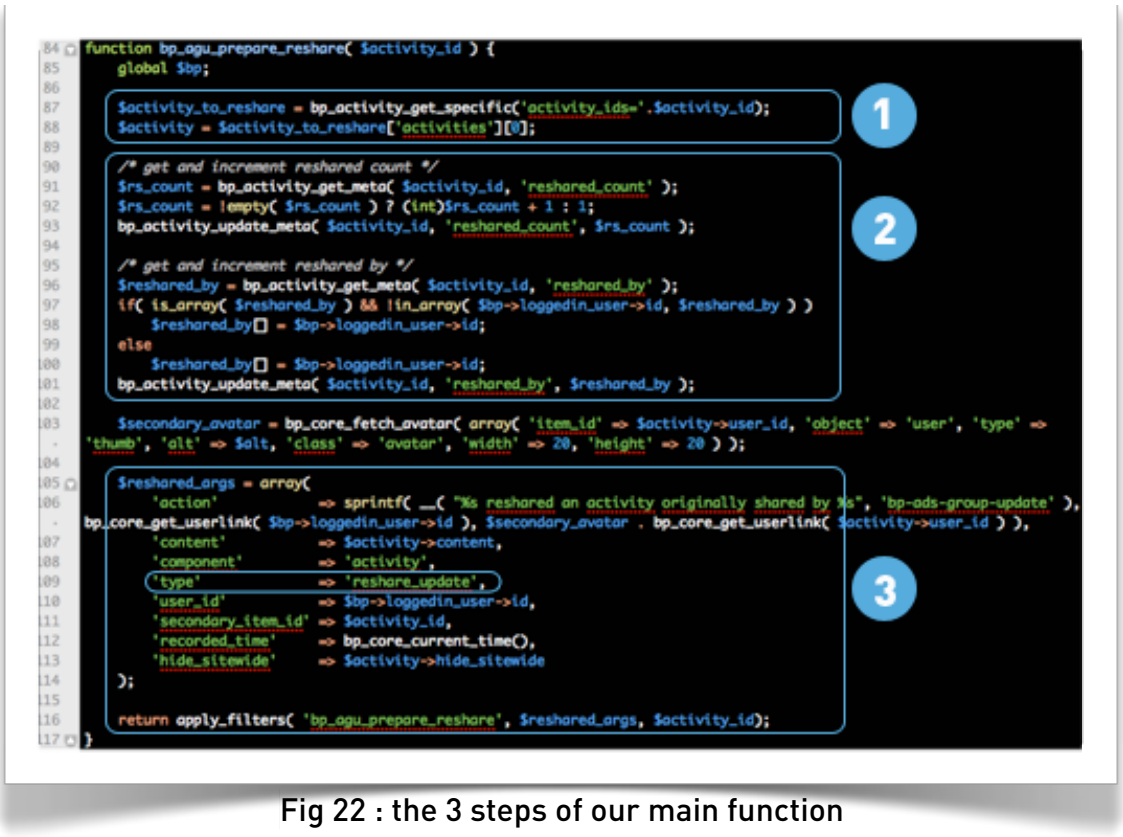


Fig 22 : the 3 steps of our main function

First, we get the elements of the activity to reshare. Its content and its visibility (**hide_sitewide**) will be used for our reshare.
Its ID will become our secondary id: this data will create the filiation link between the activity first posted and its reshare.

Step 2: we recover the meta "reshares counter" attached to the activity to be reshared (**the parent**), if it exists its counter is incremented by 1 else we set it to 1 before updating this meta.
Then let's take care of the user_ids array who "reshared" the parent activity, we add the id of the connected member if not already in this array, otherwise, if the table does not exist it is created by inserting the id of our member.
We update the second activity meta.

Finally, we build the arguments of our new activity before returning them. Did you see I added a filter if a third party plugin or a theme functions.php wants to intercept it ?
Note that the **bp_agu_prepare_reshare()** function is also called if the user has disabled its browser javascript.
In that particular case, we intercept **bp_actions** hook to check if the variable $_GET['to_reshare'] is not empty then we eventually add the activity.
We can inform the user of the success or failure of this action via the **bp_core_add_message()** fonction before redirecting the user on the same page (striped of the variables).
You will find the code of the **bp_agu_handle_nojs_reshare()** function in the plugin available for download at line 148 of the includes/bp-agu-reshare.php script.

We almost finished! Like what we did for the support message functionality, we will not forget to create the function that will remove **reshare_update** typed activities if the main activity had disappeared, and to adapt our uninstall.php to delete our fingerprints!

To In(~~finity~~)ternational and Beyond

Allow the plugin to be translated is from my point of view very important because it will maximize its use Worldwide!
WordPress uses the gettext library, you'll find in the codex all available translation functions.

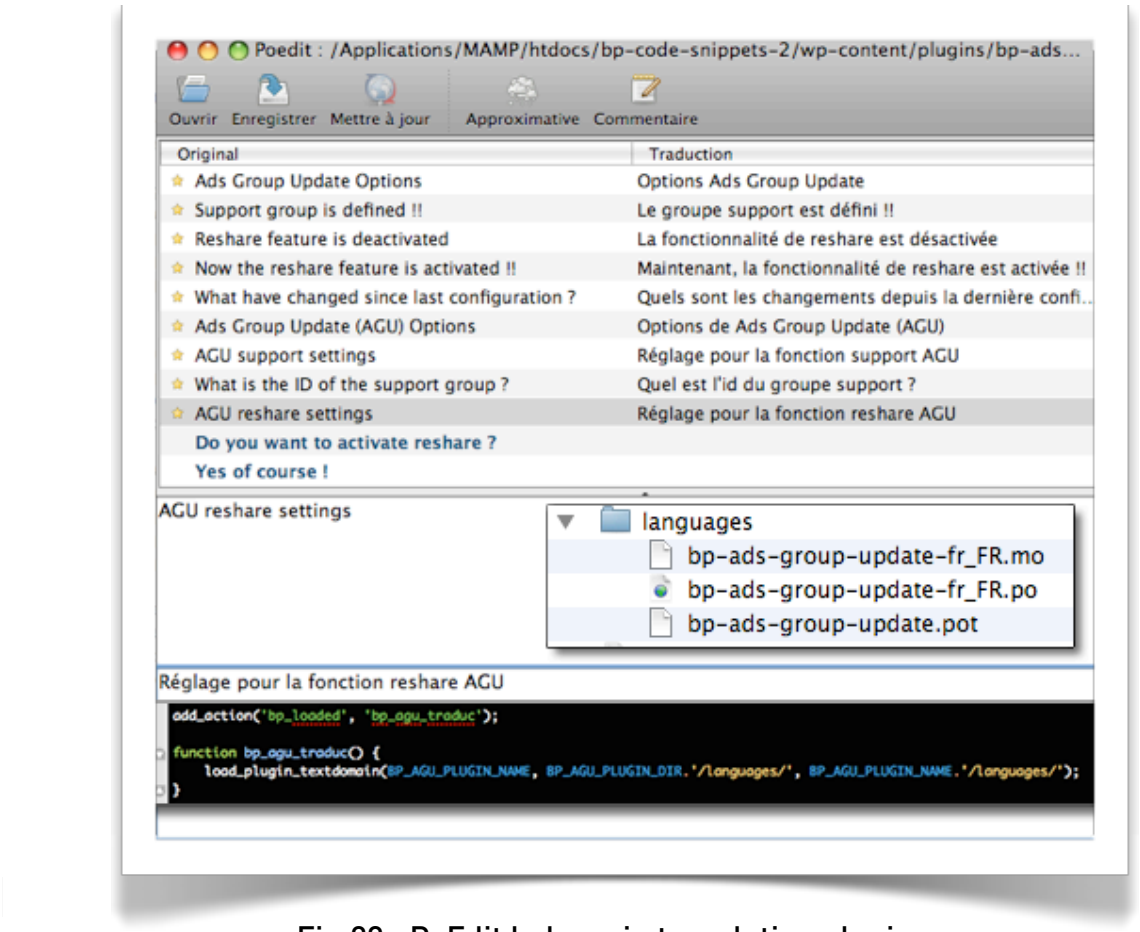


Fig.23 : PoEdit help us in translating plugins...

In our plugin, every time we wrote a message to be displayed in French, we just need to translate them into English and include them in functions such as:

- **__ ('Translate!', 'our_plugin_id')** to return the translation in a variable
- **_e ('Translate!', 'our_plugin_id')** to display the translation in the browser

We load our "plugin_textdomain" by hanging on a hook involved early enough in the loading of BuddyPress (eg: on the black background in the illustration above).
We build a .pot file that will specify the line numbers of the php scripts with the elements to be translated for each and we finish by generating language catalogs (po and mo fr_FR files) using PoEdit.

Possible evolution of the function reshare

For the purpose of the tutorial on activity metas, we store the user_ids of members who reshared an activity in a array attached to the first posted activity. However, as this

information is strongly linked to the user, it would probably be a better place to store it in user_metas. We could easily create a new area for connected members like the "My favorites" tab. It could be called "My reshares". In the user_meta of the logged in user, you would store an array of **reshare_update** typed activity ids, then you could send this array to the **bp_activity_get_specific()** function in order to list the reshares

Download the final version of our plugin!

Crédit photo : applause by Alan Wiig, on Flickr

You have arrived so far? **Huge Congratulations** to you! I recognize that this first tutorial is relatively long. I've tried to share with you as closely as possible the result of my explorations of BuddyPress. There are certainly opportunities for improvement but I think we have an interesting basis to move to a higher stage: design of a component using the **BP_Component** BuddyPress class ... Coming soon, very soon on this blog...