BuddyPress Privacy

BuddyPress Privacy Manual

Draft

Version 1.0

January 24, 2011

Compatible with BP Privacy plugin v1.0-RC1

This Manual is bundled with the plugin. Look in the /manual directory of the plugin's folder.

BuddyPress Privacy Manual

Draft Version 1.0

Table of Contents

Disclaimer

The BuddyPress Privacy Manual is offered in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

This is a draft version of the BuddyPress Privacy Manual. As such, it may contain typos, inaccuracies, and incomplete information. The manual should be used as a general guideline. Make sure you fully understand what you are doing before taking any actions that might impact the functioning of your WordPress-based network.

Proceed at your own risk. You are entirely on your own. As always, fully backup your WordPress-based network before taking any actions that could impact its functioning.

I am not responsible for any issues you have with the below suggested approaches, steps, and techniques—including ones that may have been caused by errors in this manual. Do not proceed unless you know exactly what you are doing. Whatever happens, you are on your own. You have been warned.

A. BuddyPress Privacy Overview

As a BuddyPress social network site owner, architect, designer, or developer, you know the importance of offering privacy filtering services to a site's members. You also understand that privacy has been one of the key components missing from BuddyPress.

Wouldn't it be nice if there was a way to provide users fine, granular control over who has access to which pieces of their personal data? After all, it's an essential feature of any robust social networking platform.

The BuddyPress Privacy Component (also referred to as BP Privacy, BuddyPress Authorization component, BP_Authz, or BPAz) is a BuddyPress component that does just that. It provides users with fine, granular control over who has access to which pieces of their BuddyPress-core generated personal data.

Privacy Filtering: It's All About What is Not There

Unlike most other BuddyPress plugins where it is easy to tell whether or not the plugin is functioning properly, BP Privacy is a different beast. Why is that? Well, for the vast majority of plugins, you are looking for something to appear—text, graphics, formatting changes, etcetera.

With BP Privacy, determining whether privacy filtering is working can be a difficult task as you're looking for what is not there—or more accurately for the absence of something. In other words, it removes certain items from display based on a given user's previously selected privacy settings and the identity of the viewing user.

Therefore, assessing whether this functionality is working properly is not easy nor quick. Whereas extensive and exhaustive testing has been preformed for this version of BP Privacy, you should independently verify that BP Privacy is functioning properly.

Important Note: If you are testing BP Privacy logged in as a Site Admin, then you will see all of your users' data as Site Admins are exempt from the impacts of privacy filtering. You must test BP Privacy as a regular member, not as a Site Admin.

Background: Authentication Versus Authorization

When it comes to user access of computer-based systems, access control has two subgroupings: authentication and authorization.

Authentication deals with the process of verifying that a given user is indeed who they claim to be. This is taken care of initially by the registration process and subsequently on each login by the login script.

Authorization, on the other hand, deals with verifying and managing the access rights a given authenticated user has to certain objects. This is usually accomplished through access control lists (ACLs). An ACL is a listing of what access rights, or authority, a given authenticated user has to a given object or sets of objects.

The term “auth” is often used interchangeably for authentication or authorization. But there is significant differences in meaning between these two terms. So as not to confuse people, new terminology has been created by the IT community to clearly differentiate between one or the other.

Because of this confusion, the process of authentication is now often referred to as A1, or AuthN, or simply Au. The process of authorization is now often referred to as A2, or AuthZ, or simply Az.

Since authentication must come before authorization, the A1-A2 ordinality of the terms is evident. This also explains these two alternate names of my component —BPAz and BP_Authz.

In brief, the following logic describes the BuddyPress Authorization (BPAz) Component:

1. Authentication is different than authorization. The former must come before the latter.
2. Users are the focus of social networks. They should have primacy when considering platform functionality. They are the super objects that create all content and therefore should have control over that content. Therefore, each object is created and owned by a user.
3. Only authenticated objects should have control over authorizations.
4. Users are the only object that get authenticated.

5. Therefore, Users are the only object that can set and manage authorizations.

Basic Definitions

Access Control List (ACL): a listing of what access rights a given user grants to their objects. Seen another way, it is a list of the access rights, or authority, a given user (authenticated or not) has to a given object or sets of objects owned by another authenticated user.

ACL dataset: the entire access control list (ACL) across all core BuddyPress components for a given user

ACL recordset: the subset of a user's ACL dataset that contains the access control list for a given component

ACL record: a single record from a user's ACL recordset

Access Control in BuddyPress

Typically, Access Control Lists and protocols focus on flexibly setting access rights to resources based on predefined roles (for example: site admin, group admin, moderator, editor, contributor, member, user). This type of access control is more often called, and more appropriately referred to as, role-based access control (RBAC).

Role-based AC is necessary when two or more people are involved in the management, oversight, and even ownership of specific subsets of data. This group of people then need to be jointly authorized to perform certain functions.

But BuddyPress is a user-centric platform where there are currently no-jointly owned or controlled subsets of data. This means that each object in BuddyPress has one and only one owner. Why is this? Because in BuddyPress, there are no roles offered by the Core components (with the limited exception mentioned below for groups).

So, ignoring the overall Site Administrator, in BuddyPress, there is a one-to-one relationship between a given piece of datum and a single owner of that datum. Thus the BuddyPress Privacy Component is a light-weight ACL implementation. A fully-featured RBAC-based ACL is not required.

Note: BuddyPress groups do allow for more than one assigned role (admin, moderator, member). Even so, a group cannot own itself. It is created by and administered by users. In fact, groups are largely owned by a single user—the user who created the group. This is as it should be since groups cannot be authenticated and, as was explained above, only authenticated objects (users) should be allowed to authorize access to objects.

If Groups are ever allowed to "own" themselves, then there would either need to be a way for each group to authenticate (that's not going to happen), or groups would have zero ability to set authorizations. It all comes down to a group requiring a user–a living, breathing, person–to mange its operations, in effect to own it and control it. Thus, the user as the only object that can be authenticated, is the fundamental object to which all authorization services are targeted.

Therefore, the BuddyPress privacy component (BPAz) implements a modified ACL approach that puts the focus solely on two user levels.

1. The superuser: Site Admininstrator(s) that owns the entire site
2. All other users: a site's members the own their individual accounts

User Roles in BuddyPress

Is essence, WordPress, not BuddyPress, defines user roles. BP Privacy is a component that operates within BuddyPress exclusively. So user roles are irrelevant to the operations of BPAz as BuddyPress focuses on users and not user roles. Whereas most access control list protocols focus on setting and managing permissions to resources (objects) based on user roles, BPAz is a user-centric protocol focused on letting users define and control who has access to their owned objects.

Therefore, the traditional setting of role-based privileges is not appropriate in BuddyPress. Instead, the user is the only and ultimate role. Thus the focus is on providing each user with appropriate control over their owned objects, over all the data that they create and own within a site.

From a Site Administrators standpoint, they are the superowner of the site, they are the superuser. Thus they have ultimate control over which privacy mechanisms are enabled, which ACL levels are offered, and whether privacy is even activated on their site.

From a site user's perspective, they are given a mechanism to control, to manage the viewing of their personal data by others—profile, friends list, activity streams, etcetera. Of course, this is dependent on what privacy rights the site administrator has enabled.

Relationship Between BP Privacy, BuddyPress, and WordPress

BuddyPress is a plugin suite, a layer that sits on top of WordPress. Whereas the BPAz is technically a WordPress plugin, living at the same level as BuddyPress, it is entirely dependent on BuddyPress. It is thus more accurately a BuddyPress plugin. Therefore BPAz offers privacy filtering and control to BuddyPress core-created objects only, not objects created and controlled by WordPress or other plugins that live outside of BuddyPress.

Therefore, it will not (in fact, cannot) offer privacy filtering to objects directly created or controlled by WordPress itself or even other third-party, BP-dependent plugins. Even though the Privacy API does offer some mechanisms for third-party developers to extend privacy to their plugins, it is up to each BP plugin developer to decide whether or not BPAz's privacy services can be utilized for offering privacy filtering to their plugins. In future versions of BPAz, the Privacy API may be more robust, allowing an easier privacy-filtering path to third-party plugin developers.

Note: "BuddyPress" blogs are not a true BP core-created object. For blogs, BPAz will offer only a simple show or don't show options within BuddyPress. This is because blogs are created and controlled by WordPress and not BuddyPress.

Unused Privacy Fields in BuddyPress Tables

The BuddyPress data schema currently has some unused fields in various tables that could be used for very-limited privacy control. It is not clear why these fields where included and whether there is any intention of using them in the future.

It is clear, however, that BuddyPress was not developed with privacy in mind as there is not current a facility for offering user-controllable privacy. It should be noted that if the BuddyPress core team decides to rectify that shortcoming, that it is better to mange user privacy via separate, dedicated ACL tables and not within the non-ACL BuddyPress tables. Using dedicated ACL tables allows for more

fine-grained access control and sequesters all privacy decisions into its own object.

OAuth and BP Privacy

If you are wondering why BuddyPress, or Wordpress for that matter, cannot use OAuth for authorizations, please read my article OAuth, BuddyPress, and Privacy.

B. Site Administrator's Guide

Note: *BP Privacy is a GPLed BuddyPress Component provided AS IS and at no cost. Read the license.txt and disclaimers.txt files that are distributed with the plugin for more details.*

The current, stable version of this plugin, v1.0-RC1, is a release candidate version to be used only in a development sandbox and not in a production environment. Use at your own risk. This plugin is also not being developed or supported anymore by the author. It is released to the BuddyPress community for it to be adopted and further developed.

This plugin is designed, but not guaranteed, to function properly when activated on a clean installation of the versions of PHP, WordPress, and BuddyPress indicated in the readme.txt file. It is designed, but not guaranteed, to function properly using the default BuddyPress theme(s). There are no guarantees this plugin will function with any 3rd-party plugin, 3rd-party (custom) theme, or any particular Web browser—although it does require a modern Web browser and you and your users must have javascript enabled.

The following sections are listed in no particular order of importance. They are provided as is in the hope that they help resolve a few, key issues and questions that BuddyPress Site Administrators may have when using BP Privacy.

If you are having difficulties getting BP Privacy to work, or if it had been working but stopped working when you upgraded to a newer version of WordPress or BuddyPress, then see the last item at the end of this section—xiv: Troubleshooting BP Privacy.

i. BP Privacy Administration

BP Privacy offers a number of admin-configurable settings that can be reached when logged in as a super admin. These setting can be found by navigating to the "BuddyPress > Privacy Settings" menu.

ii. Issues with Setting Admin Privacy Options

Warning: *Before making any direct changes to your database you should back it up. If you are not comfortable working with MySQL via phpMyAdmin, or some other database administration tool, then do not proceed.*

If you are having troubles setting privacy options (including not been able to see the entire privacy settings screen), then the first step to try is deleting the options entry bp_authz_admin_settings_options in your MySQL database.

Where this entry is found depends on whether you're using WordPress as a single blog install or multisite install. If using WordPress to host a single blog, then look in the wp_options table. If using WordPress in multisite mode, then look in the wp_x_options table.

If your BuddyPress install is running on a different blog ID than number 1, then you should look for the bp_authz_admin_settings_options entry in the wp_x_options table where x equals the blog ID that BuddyPress is installed under.

Once you have deleted this metadata record, you should have access to the full Admin Privacy settings screen once again. However, the settings will be reset to default, so you will have to reset them to your desired level.

iii. More About the Lockdown settings options

The lockdown settings options is for basic site-wide privacy. It does not create a perfectly private site. The BuddyPress Privacy Component is primarily targeted to offering member-configurable privacy filtering services.

To enable site lockdown, the welcome.php and maintenance.php template files need to be available to BuddyPress. This is easily done by copying the folder "/privacy" found under the "/themes" directory into the bp-default directory or your custom theme's root level. Just copy the "/privacy" folder. Do not copy the "/themes" directory.

When using this setting, you may need to implement additional measures to create an entirely private community. For instance, it is possible that any and all of BuddyPress' feeds may still be visible to the outside world.

There is a yet-to-be-implemented function, bp_privacy_block_feeds(), within the Privacy API that is hardcoded to remove all BuddyPress-generated feeds. This function is deactivated be default. You can uncomment the add_action() line to enable it but it will remove all the site's feeds for all members.

In a future version of BP Privacy, this function might be enabled so that Site Admins can have an option to set whether RSS feeds are private or not and possibly also offer individuals the right to set whether or not their content is exposed via RSS feeds. However, this feature is not currently available in v1.0-RC1.

iv. More About the Privacy Templates

The three custom templates are very simple. Within each template file are filters that you can hook into to control the basic outputted text.

You should manually edit these files, styling them to fit in with your theme's design, and augmenting them with more detailed information—for instance, adding your real privacy policy to the privacy-policy.php file. You can also replace these files with your own custom template files. As long as they have the same name as the privacy templates for which you are substituting, they work within BuddyPress, and they are located where the special privacy templates are supposed to be located, they should function fine.

v. Using MySQL's InnoDB Storage Engine for the ACL Tables

***Disclaimer**: Proceed at your own risk. You are entirely on your own. I am not responsible for any issues you have with the below suggested approaches, steps, and techniques—including ones that may have been caused by errors in the below outline. Do not proceed unless you know exactly what you are doing. Whatever happens, you are on your own. You have been warned.*

See this article to learn a little more about MySQL's InnoDB storage engine as it applies to BP Privacy.

To take advantage of the InnoDB storage engine for BP Privacy, you must manually configure the BP_AUTHZ_USE_INNODB constant. This is done by placing the following in your site's wp-config.php file (see lines 51-86 in bp-authz-core.php):

```
// Install ACL tables using the InnoDB storage engine
define( 'BP_AUTHZ_USE_INNODB', true );
```

If you've already installed BP Privacy with the default MyISAM storage engine but want to take advantage of the InnoDB storage engine instead, you have two options.

1.) The easy option. This should only be used if you have zero stored privacy data or privacy data that you do not care if it gets deleted (because it will get deleted). If this is the case, and you wish to proceed, here are the suggested steps.

a. Follow steps 1-4 of in the below section entitled "Performing a Clean Install of BP Privacy". Do not perform step 5.

b. Next, as detailed above, you have to add the code to the wp-config.php file for setting the BP_AUTHZ_USE_INNODB constant.

c. Finally, you must reinstall BP Privacy and then check your MySQL DB to make sure that the tables have now been installed to use InnoDB.

2.) The second option is to be used if you need to keep any of the privacy data currently stored in the ACL tables.

a. First, as detailed above, you have to add the code to the wp-config.php file for setting the BP_AUTHZ_USE_INNODB constant.

b. Next, you need to manually change the storage engine type for the xx_bp_authz_acl_main and xx_bp_authz_acl_lists tables from MyISAM to InnoDB using phpMyAdmin or whatever backend system you use for managing MySQL.

c. After you have done this, you'll need to manually create the proper Foreign Keys and select at least the Cascade Deletes option (I would also suggest selecting the Cascade Updates option as well—see below for the reason). In phpMyAdmin, you can do this by going to the “Structure” tab for both ACL tables and searching for the link “Relation view” near the bottom of the field listing. This link will only be visible once you set the “Storage Engine” type to InnoDB under the “Operations” tab.

Note on Cascade Updates: Although the ACL class model coding does not take advantage of the InnoDB storage engine’s ability to cascade updates–it only cascades deletes–I included that syntax in the CREATE TABLE definitions for a few reasons:

* It forces MySQL to check to make sure a parent record exists before adding a new record to a child table (never a bad idea).
* If for some reason you ever compact the main ACL table and the ID field (the Primary Keyed field) is renumbered from the start, then the corresponding fields in the child table will automatically be updated with the new ID numbers. This is more than likely something that will not be an issue, but it is a nice insurance measure nonetheless.

vi. Performing a Clean Install of BP Privacy

***Warning**: Before proceeding with this course of action, make sure you have a current backup of your WordPress MySQL database just in case.*

A clean install of BP Privacy means deleting everything associated with the BP Privacy Component. You should execute this operation in the following order:

1. Purge BP Privacy's metadata records by following instructions in the below section entitled "Resetting BP Privacy's Metadata Settings"

2. Deactivate BP Privacy if you have not yet done so

3. Delete the bp-privacy plugin from your /wp-content/-plugins directory

4. Drop the ACL tables from your MySQL database following the instructions in the below section entitled "Dropping BP Privacy's ACL Tables from MySQL"

5. Manually reinstall and then activate BP Privacy

vii. Resetting BP Privacy's Metadata Settings

To reset BP Privacy's metadata settings, you need to purge a few metadata records from WordPress' meta tables (the wp_sitemeta and wp_x_options tables if using multisite). You can easily do this by uncommenting a few lines of code in the function bp_authz_plugin_deactivated() found in the file bp-authz-loader.php.

These lines are commented out by default as indicated in that function's inline comments:

> "If you enable the code below, then you will have to reset any previously disabled Admin privacy objects. In other words, when you reactivate the privacy component, it will run with all privacy objects fully active. By keeping this code disabled, you can easily deactivate privacy filtering and then reactivate it without losing any previous settings."

Simply uncomment those lines before deactivating the Privacy Component (plugin). You must then deactivate BP Privacy, triggering the newly-uncommented code, to purge the privacy metadata records.

At this stage, the component will be reset to it's default metadata settings. If you plan to reactivate the component right after this step (meaning you are not performing a clean install as detailed above), then make sure to comment back the aforementioned code lines so that the settings do not get purged when you deactivate the plugin the next time.

A future version of the Admin Menu may allow you to click a box to indicate whether you wish BP Privacy metadata to be purged from WordPress' metadata tables automatically upon plugin deactivation.

viii. Dropping BP Privacy's ACL Tables from MySQL

***Warning**: It is always wise to fully backup your database before performing any database (DB) management task. This backup may prove indispensable if you make a mistake when working in the DB's back-end environment.*

A clean install requires that you remove the ACL tables from your MySQL database. In DB parlance, this is called dropping the tables from the database.

BP Privacy's ACL tables are bp_authz_acl_main and bp_authz_acl_lists (both of these tables will more than likely have a "wp_" prefix).

If you've installed the two ACL tables to use the InnoDB storage engine, then you will have to drop the child table xx_bp_authz_acl_lists before the parent table xx_bp_authz_acl_main. Attempting to do it the other way around will not work. You will get a warning message.

Only do this if you absolutely do not care about any data that is stored in the ACL tables as once those tables are dropped, all data will be lost.

ix. Sharding Your Database

As the privacy ACL object created by BPAz's two classes do require joins, if you are sharding your database, it is best to keep both xx_bp_authz_acl_main and xx_bp_authz_acl_lists tables on the same partition.

x. Issues with BP Privacy Making AJAX Calls When the wp-admin Directory is Password Protected

BP Privacy declares its own JavaScript namespace object for handling all in-plugin AJAX requests. If you have added server-side password protection to

your /wp-admin directory, then any AJAX requests will fail. This is because the key file that handles all AJAX requests in WordPress is the wp-admin/admin-ajax.php file.

Since BP Privacy, and all other 3rd-party plugins that might make AJAX requests, can't access the /wp-admin directory without a password, they will not be able to utilize AJAX.

If you have password protected your /wp-admin directory, then you will need to turn off the "Members of these Groups" and "These Users Only" ACL Privacy Settings in the Site Admin Privacy Settings page.

See the subsection entitled "Using AJAX to display Group and User Listboxes" in the Developer's Guide (Section D below) for more details.

xi. Consider Disabling "Members of these Groups" and "These Users Only" ACL Privacy Settings For Large Sites

Although BP Privacy offers a number of increasingly fine user-access privacy settings, for large sites (greater than 500 to 1000 members), it may make sense to disable the "These Users Only" ACL Privacy Setting in the Site Admin Privacy Settings page.

For sites that have more than 250 groups it might also make sense to disable the "Members of these Groups" ACL Privacy Setting.

For both of these cases, you will have to determine what "large" actually means from your users' perspective. Is more than 100 users too large to make use of the "These Users Only" listbox worthwhile? Is it 250? Is it 750? At what point does the total group count become too large to make use of the "Members of these Groups" listbox worthwhile? Each niche-network will have its own sweet spot with regards to these two issues.

xii. A Note About the “Members of These Groups” and “These Users Only” ACL Options

The ACL options “Members of These Groups” and “These Users only” use JSON to populate the listboxes. You must be running at least PHP 5.2.0 for this to function properly. Furthermore, the character collation setting in your

wp_config.php file must be set to UTF-8 as the PHP json_encode() function only works with UTF-8 encoded data.

This is how it should look in your wp-config.php file:

```
define('DB_CHARSET', 'utf8');
```

Therefore, if you are running a PHP version older than 5.2.0, or your character collation is set to something other than “utf8”, then you will need to disable the “Members of These Groups” and “These Users only” ACL options in the Site Administrator’s backend under “BuddyPress > Privacy Settings > Customize ACL Settings”.

xiii. Issues Filtering Legacy Activity Stream Items

If you have been running your BuddyPress-powered community for more than a year, there will be a few activity types on which your site’s users will not be able to set activity privacy filtering. For instance, BuddyPress used to have a component called the Wire. It also has renamed a few activity items over time and those items as well will cannot be filtered out.

Only BuddyPress’ current activity items can be filtered by BP Privacy.

xiv. Troubleshooting BP Privacy

Warning: *Before making any direct changes to your database or our site’s directory structure, you should completely backup your database and site. If you are not comfortable working with MySQL via phpMyAdmin, or some other database administration tool, then do not proceed. If you are not comfortable with any of the below procedures, do not proceed. You are entirely on your own. Proceed at your own risk.*

a. Issues with initial installation

If you are having issues getting BP Privacy to install, please make sure that WordPress and BuddyPress are functioning properly. If so, then make sure that you are using the versions of PHP, WordPress, and BuddyPress required (as listed in the plugin’s readme.txt file).

If all of the above check out, then you may need to reinstall BP Privacy. To do that, you should deactivate BP Privacy if it is activated, then delete the /bp-privacy folder from the /wp-content/plugins directory. Next, manually reinstall BP Privacy and then try reactivating.

If you are still having issues, see the next item.

b. Issues with 3rd-party plugins and custom themes

As stated in the introduction of this section, BP Privacy is designed, but not guaranteed, to function properly using the default BuddyPress theme(s). There are no guarantees this plugin will function with any 3rd-party plugin, 3rd-party (custom) theme, or any particular Web browser—although it does require a modern Web browser and you and your users must have javascript enabled.

If you are still having issues getting BP Privacy to function properly, you may have a plugin conflict, an issue with the theme you're using (if it is not the bp-default theme), of some other error such as some incompatibility being caused by a code customization on your site.

What you need to do is distill your site's operating environment down to the least common denominator.

The first step is to make sure that you are using BuddyPress' default theme. Switch to it and see if the issue goes away. If it does, then there is some problem with your custom theme. Contact the designer of the theme.

If the problem does not go away, then it is time to focus on possible 3rd-party plugin conflicts. There are two approaches. The first is to deactivate one plugin at a time, testing after each deactivation to see if BP Privacy starts functioning properly again. If it does, then the last plugin you deactivated is likely the cause of the conflict.

The second approach is to deactivate all 3rd-party plugins except BP Privacy and see if BP Privacy starts functioning properly again. If it does, then the probability of a 3rd-party plugin conflict is high. You then

need to reactivate plugins one at a time, testing after each reactivation until the problem returns. When the problem returns, then the last plugin you reactivated is likely the cause of the conflict.

It can also often be useful to check your server's error logs, in particular the log for PHP errors. Any errors in that log file can shed light on what is happening with your site's operating environment.

c. Issues after upgrading WordPress or BuddyPress

It is likely that when WordPress is updated to the 3.1 series and BuddyPress is updated to its 1.3 series, that BP Privacy will have a number of issues and stop working. It will not only be BP Privacy but many plugins and even themes that will have issues.

BP Privacy has gone through two major code refactorings as a result of changes to BuddyPress' core codebase. It happened when BuddyPress went from the 1.0 series to the 1.1 series. It happened again when BuddyPress went from the 1.1 series to the 1.2 series.

It is not unusual for the core development team to change functions that plugins rely upon. Even a minor change in a function's behavior can cause a plugin to stop working. Sometimes the changes are more major like restructuring key object arrays that a plugin might require to perform some function. There are a number of minor to major changes that can occur between major version releases.

With BP Privacy, BuddyPress core code changes can have more of an impact because BP Privacy is tightly tied into core functionality. It is a plugin that alters the way the core outputs data. So, unlike other 3rd-party plugins, BP Privacy is highly-dependent upon deeply-connected, and rarely used (by other plugins) object arrays.

With a major version upgrade of BuddyPress, tracking down all the BuddyPress-core codebase changes that are impacting BP Privacy's operation is not an easy or quick process. The first place to start is to examine the PHP error log. After that, it can be a slow and difficult process requiring a function by function check (within BP Privacy), close reexamination of the contents of the various BuddyPress-generated

object arrays that BP Privacy relies upon, and sleuthing around in general. Sometimes something as simple as a renamed, deprecated, our re-prioritized BuddyPress core hook can be the root of a problem. But, more than likely, it will not be one problem. It will be a number of core codebase changes that are causing the issues.

Remember, BP Privacy is a 3rd-party plugin. It is not a core BuddyPress component. Privacy is an after thought, if even that, in BuddyPress. So a lot of bending, prodding, and jury-rigging were required to get BP Privacy to integrate into BuddyPress in the first place. Expect many tens of hours, possibly hundreds of hours, retrofitting BP Privacy to get it to function properly again in each newest version of BuddyPress (and sometime also WordPress).

d. General issues and support

As mentioned in numerous places in this document, the BP Privacy plugin package, and a number of blog postings, BP Privacy is not being developed or supported anymore by the author. It is released to the BuddyPress community for it to be adopted and further developed.

Therefore, if you have issues or questions that are not answered here or that you cannot figure out on your own, you should start a thread on the BuddyPress Support forums and see if anyone there can offer support.

It may be necessary for you to hire a skilled BuddyPress developer to help troubleshoot the issue. I am not for hire so please do not contact me. Also, I will not be answering any email about BP Privacy, including requests for my suggestions on competent developers to hire. Please use the BuddyPress Support forums instead.

C. Site User's Guide

Note: *BP Privacy is a GPLed BuddyPress Component provided AS IS and at no cost. Read the license.txt and disclaimers.txt files that are distributed with the plugin for more details.*

[This section needs to be completed]

Saving Global or Group-level ACL Settings on Tiered Privacy Forms: To successfully save global or group-level ACL settings, you must check the “Apply Globally on Save” or “Apply to Group on Save” checkboxes respectively. If you do not, your selected settings will not be saved.

Resetting ACL Levels on Tiered Privacy Forms: On tiered privacy settings forms, it is easy to reset the ACL levels back to default (meaning back to "All Users") for all the privacy items on the given form. This is also called zeroing out your privacy settings.

This allows users a unique way to easily and quickly reset all ACL levels across these components, or just a subset of the ACLs within a group, back to default. Currently for BuddyPress' core components, multi-tiered privacy settings forms are available for profile and activity items only.

To zero out your ACL levels across an entire component, you choose the "All Users" settings in the Global "Who Can View" dropdown box and then make sure that the "Apply Globally on Save" checkbox is checked. Hit the "Save Changes" button at the bottom of the form and all of your privacy items for the given component will be reset to default.

A similar process applies to zeroing out the privacy items within a group. Of course, you can also zero out a single ACL setting on its own. The method described above is another example of the flexibility and control offered to users via BP Privacy.

D. Developer's Guide

Note: *BP Privacy is a GPLed BuddyPress Component provided AS IS and at no cost. Read the license.txt and disclaimers.txt files that are distributed with the plugin for more details.*

1. General steps to add privacy to your custom plugin:

a. First read the Developer's Guide in full to get a general understanding of some of the key programmatic features of BP Privacy.
b. Register your plugin with the BP Privacy Component: this will add your plugin to the list of privacy services that the Site Admin can enable or disable.
c. Create your custom privacy settings screen(s): you'll need to create your own loops that grab the appropriate data from your plugin's table(s)/ array(s) at the appropriate time. Look at BP Privacy's settings files for examples of both tiered and single settings forms.
d. Add your component's privacy settings to BP Privacy's menu
e. Create custom filter(s): grab any user ACL preferences for the data that your plugin stores in the ACL tables and use that data to perform whatever filtering is required. At the start of your privacy filtering routines, make sure to check whether privacy filtering is enabled for your component. See the BP Privacy's core filters to see how it is done.
f. To have BP Privacy automatically take care of filtering activity items generated by your plugin, make sure that activity stream actions are properly registered with BuddyPress. Use bp_activity_set_action() and bp_activity_add() to do this. Search BP core to see how the core components utilize these functions.

2. Creating a Tiered Privacy Settings Screen

In a future version of BP Privacy, there may be a developers directory that holds bare-bones templates for creating tiered and single privacy settings forms. But for now, you need to carefully study the privacy settings forms that BP Privacy uses to understand how to create and implement your own privacy settings form(s) for your custom component.

i. Auto triggering tiered containers: The Global Groups Singles Grid Array

Note: This is automatically handled for you if you use the suggested coding within the example tiered privacy settings form.

This section needs to be written. For some details, read the “Populate $containers_to_trigger array” and “Looping Through the $containers_to_trigger Array” comment blocks within any tiered settings form.

ii. Using AJAX to display Group and User Listboxes

a. BP Privacy's JavaScript namespace object

BuddyPress declares its own AJAX namespace so themes can access the WordPress AJAX functionality (see bp_core_add_ajax_url_js() in bp-core-cssjs.php). But BP Privacy is a plugin, not a theme. Relying on wp-load.php for plugin AJAX functionality is not a good idea.

So, BP Privacy creates it's own JavaScript namespace object for the file that handles the AJAX request. It does this by directing all AJAX requests to wp-admin/admin-ajax.php. The admin-ajax.php file processes both admin AJAX requests as well as front-end requests. See Note 2 on this WordPress Codex article entitled AJAX in Plugins to learn more about this overall issue.

This new AJAX namespace is created in bp-authz-cssjs.php in the bp_authz_create_js_namespace() function.

b. AJAX Calls in BP Privacy When the /wp-admin Directory is Password Protected

See the subsection entitled "Issues with BP Privacy Making AJAX Calls When the wp-admin Directory is Password Protected" in the Site Administrator's Guide (Section B above) for more details.

c. Outputting Group and User Listbox Data

Note: This is automatically handled for you if you use the suggested coding within the example tiered privacy settings form.

Upon form load of any privacy settings screen, if the BPAz level of a given privacy item is currently set to 3 or 4, then the function that outputs the group and user listboxes (bp_authz_create_privacy_settings_listbox() function within the BP Privacy API) is called to populate the screen with the appropriate data.

However, if the BPAz level is something other than 3 or 4, the listbox function is not called. This is handled this way to limit the number of resources called on form load. There is no reason to pre-populate certain form fields if they are not currently being used.

Whenever the ACL selector is changed, it triggers the jQuery change() function. If the newly-set ACL level is 3 or 4–which means that the user has chosen either "Members of These Groups" or "These Users Only" in the "Who Can View" selector–then a series of events transpires which leads to the triggering of an AJAX request. This AJAX request will autogenerate and output the contents of the appropriate listbox.

Note: in the above scenario, the BPAz level is not currently set to 3 or 4 for that privacy item as if it was, the listbox would be displayed on form load.

However, for the jQuery function to work, specific data needs to be passed into it. This data is available via several hidden input fields within the divisional id element. The divisional id selector containing the required hidden variables has a postfix of "...-listbox", is unique and depends upon which privacy item is currently being acted upon. When the ACL level has been changed to 3 or 4, the jQuery function will fire the serializeArray() command which grabs these prepopulated variables from the hidden input elements.

Once the data has been grabbed by the jQuery function, it is passed as parameters into bp-authz-listbox-ajax.php. This file uses the bp_authz_create_privacy_settings_listbox() function within the BP Privacy API.

The result returned from the listbox function is a html-formatted string that gets passed back to the calling jQuery function via a JSON-formatted array. If the procedure indicates success, then the jQuery

function will output the passed-in html string into the underlying divisional container . This creates the desired select box with any previously selected options highlighted.

This is why BP Privacy requires PHP 5.2 at a minimum as the json_encode() function is not available in prior versions of PHP.

Note on passing the lists array into jQuery using JSON: As the lists array of the current privacy item object is an associative array by design, this causes an issue with its usability within jQuery as javascript does not support associative arrays. Therefore, the elements within the lists array need to be converted to a usable key-value pair-based string before being used by jQuery and before being passed into the bp-authz-listbox-ajax.php(). This is accomplished via the json_encode() function.

d. The AJAX spinner

As the above detailed procedure can result in an AJAX event being triggered, server response time my vary. Therefore, it is wise to provide some visual cue to the user that something is indeed happening and that they should be patient (hopefully that means only for a second of two at most). The best visual cue is to display an animated "loading" image. That's where the ajax-loader.gif file located in the components / images directory comes in.

When the form is first rendered, the AJAX spinner image (ajax-loader.gif) is populated but hidden via it's class selector "ajax_spinner" which defaults to display:none in the CSS file. If a user requests to see the group or user listbox (BPAz 3 or 4), it triggers the jQuery sequence and the AJAX spinner is displayed while the request is being processed. Once it completes, the image is hidden once again.

You might be thinking that this UX feature is easy to implement. I certainly thought that would be the case. But after trying all the standard methods of showing and hiding an AJAX spinner image, it become clear that this was not going to be an easy challenge to figure out.

Why? Well, as I studied this, I realized that most applications have a much simpler need for displaying such a visual clue. Most just have one

cue, not the many possible cues that populate the tiered privacy form—one for each privacy item listbox.

What happens using the traditional methods? For each privacy item where the listbox has been previously selected, the next time the user chooses to view the listbox for another privacy item, all of the animated AJAX loading spinners are toggled on then off. That's correct—*all of them at once.* The reason for this, as well as the solution, is described below.

I first tired to use the below standard method of showing then hiding the AJAX spinner.

```
jQuery("div#" + idname + "-listbox img.ajax_spinner")
	.ajaxStart(function(){
		jQuery(this).show();
	}
	.ajaxStop(function(){
		jQuery(this).hide();
	}
);
```

But these AJAX event handlers are global. They cause the image to be shown or hidden for all previously-triggered AJAX events. This is due to the way jQuery handles global AJAX events and how these AJAX events are stored in the browser's cache. So this did not work.

I then tried experimenting with trying to use local-level AJAX events. This failed as well. The beforeSend function would not trigger. Only the complete function triggered.

```
jQuery.post(PrivacyAjax, {
	beforeSend: function () {
		jQuery("div#" + idname + "-listbox img.ajax_spinner").show
();
	},
	complete: function () {
		jQuery("div#" + idname + "-listbox img.ajax_spinner").hide
();
	},
```

```
        action: "bp_authz_ajax_listbox",
        '_wpnonce': ""+wp_nonce_name+"",
        ...
});
```

I then experimented with setting the cache and global settings to false for each AJAX post request. That did not work. Numerous permutations of CSS, JS, and AJAX did not make a difference.

Finally, I hit upon what is a rather brute force way to control the display of the AJAX loading spinner just for the privacy item in question—via insertion of the element into the DOM with jQuery's .append() command after the listbox html was outputted.

When a user chooses to display a listbox, other than one that may already be displayed upon form load, the jQuery event will trigger the display of the AJAX loading spinner. Once the AJAX request is finished, assuming success, the contents of the division that encloses the listbox is rewritten. Since the AJAX spinner gif is included within that division, it is effectively removed and its display is stopped.

But this means that the next time a user chooses to display a listbox, the animated Ajax-loading-spinner image cannot be displayed since the image element containing it no longer exists. The way to solve this is to make sure that the image element is added back to the division after output of the listbox. This is where the append function comes in handy.

See http://docs.jquery.com/Ajax_Events for more details about global versus local AJAX events and http://api.jquery.com/jQuery.ajax/ for details on various AJAX settings.

3. Processing Privacy Settings Form Data

Data from each privacy settings form is stored in the bp-authz $_POST array. This array is a multidimensional array whose stored data format depends on the type of privacy settings form being used: tiered or single.

Tiered privacy forms provide a multi-level privacy settings view that allows users to apply a given ACL setting globally, by group, or by individual item

(single). Single privacy forms contain the data for a simple list of individual privacy items.

What is a tiered privacy settings form? A tiered privacy from contains three sections:

1. Global: apply selected ACL setting across the ACL recordset
2. Group: apply selected ACL setting to all privacy items in a given privacy group (i.e., Field Group, Activity Group)
3. Single: apply selected ACL setting to a single privacy item (i.e., field ID, specific activity of an Activity Group, “Who can request friendship” item, etcetera)

a. Overview of fields sent to $_POST array

[id] = unique ID; populated from ACL Main table record ID or null if new record

[filtered_component] = name of BuddyPress core component to which ACL settings apply

[filtered_item] = For xprofile data: profile_global | profile_group | profile_field; for activity data: activity_global | activity_group | activity_field; for all other components, this is value depends on the privacy item.

[item_id] = The identifiable ID associated with item; as an example, a xprofile field ID or xprofile group ID; for all other BuddyPress core components, this issue becomes clouded. See subsection d below for more details (creating a Unique Item ID When BuddyPress Does Not Offer One).

[acl] = the ACL (privacy level) to apply to given privacy item

For Global only:
[save_global] = boolean to indicate whether ACL settings should be applied across ACL recordset

For Groups only:
[save_group] = boolean to indicate whether ACL settings should be applied across given group

For Single items only: **** Or is this for all items?
[keep_list]

b. Tiered Privacy Form Data Array Structure and Array Element Levels

You might be wondering why group elements in the multi-tiered array are positioned at the same level as the global element (both at Level 1). The reason is that for a given core BuddyPress component, there is no benefit in simultaneously storing a global privacy record and a group privacy record in the ACL tables.

If a user has indicated that a given ACL setting should be applied globally across an ACL recordset, then that implies that the global ACL setting should also apply to each group as well. If a user has selected an ACL setting to be applied within a group, then that implies that the global setting is not to be applied. Thus, there is never a case when both global and group ACL settings should coexist in an ACL recordset for the same component for a given user.

However, global and group records are used internally to BP Privacy for the sole purpose of offering users multi-tiered privacy settings screens. The stored values are not used in any of the privacy filtering routines as BuddyPress does not recognize these data as core-generated data. This means that in order for the user to derive value from setting a global- or group-level ACL, the data for each individual field–found within each ['singles'] array–must actually be saved as well.

For instance, the Field ID of each xprofile field or the activity type within an activity type group must be stored in the ACL recordset irrespective of whether or not a user has indicated that a global or group ACL should be applied. All of this complex code looping is solely for the benefit of providing multi-tiered privacy settings functionality.

Multi-tiered Privacy Form Data Array Structure and Array Element Levels

Level 1:
['global']

Level 2:
['id']
['filtered_component']
['filtered_item']
['item_id']
['group_user_list_old']
['acl']
['grouplist']
--> reserved for later use ['keep_grouplist']
['userlist']
--> reserved for later use ['keep_userlist']
['save_global']

Level 1:
['groups']

Level 2:
['group-###']

Level 3:
['id']
['filtered_component']
['filtered_item']
['item_id']
['group_user_list_old']
['acl']
['grouplist']
--> reserved for later use ['keep_grouplist']
['userlist']
--> reserved for later use ['keep_userlist']
['save_group']
['singles']

Level 4:
['single-###']

Level 5:
['id']
['filtered_component']

['filtered_item']
['item_id']
['group_user_list_old']
['acl']
['grouplist']
--> reserved for later use ['keep_grouplist']
['userlist']
--> reserved for later use ['keep_userlist']

c. Single Privacy Form Data Array Structure and Array Element Levels

Level 1:
['singles']

Level 2:
['single-###']

Level 3:
['id']
['filtered_component']
['filtered_item']
['item_id']
['group_user_list_old']
['acl']
['grouplist']
--> reserved for later use ['keep_grouplist']
['userlist']
--> reserved for later use ['keep_userlist']

d. Creating a Unique Item ID When BuddyPress Does Not Offer One

Unfortunately, there are a number of instances where BuddyPress stores data that is not easy for a developer to find. Instead of using a lookup table, key/value pairs are used to store datasets in object arrays that are manually rebuffered into memory each time a page is loaded.

One such case is that of activity actions. These get stored in memory in the $bp->activity->actions object array subelement. Because of this, BP Privacy has to attempt to create a unique identifier for each activity so that it can properly identify it in the main ACL table. Unlike the Xprofile component that offers unique field group and field IDs be virtue the auto-incremented ID field, when it comes to privacy, most other components do not offer unique numeric identifiers that a developer can count on never changing.

This is not a desirable way in which to handle this issue but as BuddyPress was not designed with privacy as a foundational element, we have to bend and retrofit the exposed data that BuddyPress offers to fit the needs of the privacy codebase as best as possible.

One such “bending” is that of creating unique Item IDs for BuddyPress core datum that does not offer any. This is seen mainly in the tiered activity settings form where a unique item ID is manufactured for activity groupings. Without a unique item ID, group-level privacy filtering would not be possible for activity actions.

For the field level, or single privacy item level, when can get away with using the value zero in the item ID field of the main ACL table as the filtered_item value can serve as a surrogate unique identifier. This of course is not desirable as it is very possible (and even likely) that in some point in the future the BuddyPress core developers may change the text of any of the values that get stored in the filtered_item field of main ACL table.

This would result in significant issues, causing BP Privacy to not be able to locate the proper data in the future, in essence causing orphaned records. This is another reason why it is crucial to offer unique identifiers. With a unique identifier based off of a table’s auto-incremented ID field, it would be more unlikely that data would become orphaned by changes made to to BP core’s values.

Let’s look out how we attempt to create a unique item ID for group-level activities. It is important to note that this technique is not fool proof. It is possible, but unlikely, that the generated item ID will not be unique. As stated above, it is also possible that in future versions of BuddyPress, the core development team may change the name of an

action group. If that happens, then this technique will fail the next time the Activity settings screen is loaded. But, we have no choice but to attempt to create a unique item ID.

You could ask why we do not just store the text-based name of the activity group as the unique item ID. We could, but the same issues would remain. If the activity name is changed, then the uniqueness of the item ID is lost and the underlying ACL data gets orphaned. So, instead of requiring the storing of alphanumeric data in the item ID field, we stick to using the more useful numeric data type.

Okay, to attempt to create a unique item ID for the activity group, we first take a subset of the alpha characters and generate an ASCII-based integer. We will be sampling three characters so as to limit the overall size of the ASCII integer. As alpha characters have ASCII codes ranging from 65 to 122, we have to make allowances for the generated integer to be up to 9 digits long. In other words, although not likely, the three characters could be “zzz” which would result in our artificial integer equaling “122122122”. Therefore, the numeric field type for the item_id field in the main ACL table is set to INT type to allow for this much larger integer needed just for this purpose.

This technique is also used for creating unique item IDs for individual activity actions and for some other BuddyPress items in other core components. Based on the array text available with which to manufacture artificial item IDs, the above detailed varies—as can be seen with the procedure to create unique item IDs for individual activity action items.

All of this code gymnastics could be done away with if BuddyPress used lookup tables for select key pieces of core data instead of storing that data in temporary object arrays.

4. Querying the ACL Tables

There are currently five different queries offered for extracting data from BP Privacy's ACL tables. They are found in the bp-authz-classes.php file.

The first four are methods of the BP_Authz_ACL_Main class. Currently, only the fourth one is used although the third one has been fully tested and works

as expected. The first two are also not used but need testing to verify that they return the expected set:

i. get_user_acl_dataset() returns the entire ACL dataset (across all core components) for a given user

ii. get_user_acl_recordset_by_component() returns the ACL recordset for a given component for a given user

iii. get_user_acl_privacy_item_by_id() returns one ACL record for a given component for a given user (record id known)

iv. get_user_acl_privacy_item_no_id() returns one ACL record for a given component for a given user (record id not known)

The last query is a method of the BP_Authz_ACL_Lists class. It is called indirectly by the get_user_acl_privacy_item_no_id() method.

v. get_user_group_lists_by_id() method: returns all ACL Lists records for a given privacy item of a given component for a given user using the Main ACL id

5. Saving and Deleting ACL Lists data

Note: See "Sharding Your Database" in Section B, Site Administrator's Guide, for important information.

Although the BP Privacy codebase takes care of the complex logic required to save ACL lists data, as a developer, you need to understand what is taking place and why.

The ACL Lists table is a normalized table that shares a many to one relationship with the ACL Main table. The association is made via the relationship between the id field in the ACL Main table with the id_main field in the ACL Lists table. This means that each record in the ACL Lists table is associated with exactly one record in the ACL Main table. However, each record in the ACL Main table can have many records that it is associated with in the the ACL Lists table.

This is a simple table-to-table relationship that RDBMSs are designed to model. It is also called a parent-child relationship. But the simplicity stops there.

Most RDBMS-powered systems have their own unique business logic that enables tabular relationships to be easily created, updated, and deleted. However, WordPress and BuddyPress do not offer any out-of-the-box relational business logic routines. Therefore, BP Privacy encodes its own.

The first crucial routine that enables BP Privacy to handle the one-to-many relationship between the ACL Main table and the ACL Lists table is the creation of the ['lists'] array subelement returned when the DB is queried during the settings screen form load (currently via the get_user_acl_privacy_item_no_id() method). Within that query, an object array is built that associates each returned list record's id with its value. Here's how the array is built for any group list records. It uses the following array composition:

```
$list_items['lists']['grouplist'][$rec_id] = $acl_lists[$i]['user_group_id'];
```

The same logic is used for user list records, substituting the 'grouplists' element name with the 'userlists' name.

This creates a 'lists' parent array that can contain up to two child arrays--one for 'grouplists' and another for 'userlists'. Within each of those child arrays, the key value pairs provide the essential information. The key is the ACL Lists record id--the unique primary key--for the given list value. This is used if a lists record needs to be deleted. So, we have this key value pairing:

```
[id] => ['user_group_id'];
```

In order for the lists table to be properly managed, two pieces of data are required: the state of the the subset of the ACL Lists table for a given ACL Main record when the settings screen was first rendered, and the state of the lists $_POST array buffer after the user hits the "Save Changes" button. These two, potentially different states, are stored in two different arrays. These two array sets are passed into the BP_Authz_ACL_Main::save() method and are used to determine how each

list record should be handled. Are we inserting a new list record or deleting a list record?

This is how the two passed-in lists arrays come into play. Using the array_diff() function, we can compare the new array element set stored in the object $this->group_user_list_id_array, with the old lists array element set stored in the object $this->old_lists_array. This will tell us the differences between the old lists state and the new one, if any. The results will help determine what action should be performed based on this simple table:

In Old Array	In New Array	Action
True	True	Do nothing
True	False	Delete
False	True	Insert

With ACL lists records, technically there is no need for updating. Therefore, there are only two actions that can be taken: delete and insert new. The creation of a new lists record is indicated when an id exists in the new lists array set but does not exists in the old lists array set. A list record is marked for deletion by a user deselecting it within a given listbox. The record id will then not be in new lists array set but will exist in the old lists array set. If a list id exists in both array sets, then no action is required. That array element can be skipped.

We need to go through this seemingly-convoluted process to determine what action should be taken, in particular if a list record needs to be deleted as the user has deselected a user or group item from a list. If the action to be taken is deletion of a list record, there is no way to know which list record needs to be deleted if we don't have the record ID from the ACL Lists table.

Fortunately, we do have the ACL list record ID. It is in the old lists array that was assembled when the DB was originally queried. So if the action equals delete, we grab the list record ID from the old array and pass that into the BP_Authz_ACL_Lists::delete_by_id() method. Remember, as detailed above, the old lists record id is stored as the element key.

6. A Note About the “Members of These Groups” and “These Users Only” ACL Options

The ACL options “Members of These Groups” and “These Users only” use JSON to populate the listboxes. You must be running at least PHP 5.2.0 for this to function properly. Furthermore, the character collation setting in your wp_config.php file must be set to UTF-8 as the PHP json_encode() function only works with UTF-8 encoded data.

This is how it should look in your wp-config.php file:

```
define('DB_CHARSET', 'utf8');
```

Therefore, if you are running a PHP version older than 5.2.0, or your character collation is set to something other than “utf8”, then you will need to disable the “Members of These Groups” and “These Users only” ACL options in the Site Administrator’s backend under “BuddyPress > Privacy Settings > Customize ACL Settings”.