Bluetrait 2 Plug-in Guide

Document Revision: 16
Date: 23-MAY-2009

Bluetrait Revision: 2.0
Release Date: 23-MAY-2009

Table of Contents

Introduction

This guide is aimed at developers who wish to extend the functionality of Bluetrait 2 through the use of plug-ins.

Plug-ins are pieces of code that are separate to Bluetrait 2 that communicate with it through the use of its API.

This allows developers to extend and add functionality to Bluetrait 2 without the need to modify core files.

Plug-ins can be as small as two files that are uploaded to the Bluetrait 2 folder. The end user is then able to activate these plug-ins through the Admin Panel.

An example of a plug-in would be the default one that comes included with Bluetrait 2; the contact form.

The contact form is a simple plug-in that adds a way for visitors of the site to get in contact with the writer of the blog.

The contact form plug-in will be talked about in greater depth later in this guide.

If you find any errors or sections of the document that are not clear please contact me:

Michael Dale
mdale@dalegroup.net

Why Plug-ins?

Q: Why would you write a plug-in?

A: To add extra functionality to Bluetrait 2.

Q: Why wouldn't you just put all the functionality into Bluetrait 2 to start with?

A:
1 – Because we don't know what everyone wants.
2 – Because we don't want to bloat Bluetrait 2 with functionality that will only be used by a subset of users.

Writing Your First Plug-in

The first step is to understand how a plug-in can interact with Bluetrait 2.

There are three main ways a plug-in can "talk to" Bluetrait:

1. Through the use of a section hook/task
2. Through the use of a content filter
3. By using built in Bluetrait functions

A section hook or task allows a plug-in to insert a function that Bluetrait runs (such as an extra spam filtering function).

A content filter is a way for a plug-in to modify text (for example to do some processing on the post body)

And built in functions (such as adding users etc)

Plugin Files and Locations

Plug-ins are stored in the following folder:

```
BT_CONTENT . '/bt-plugins/'
```

This defaults to:

```
bt-content/bt-plugins/
```

Please note that BT_CONTENT can be changed by the end user.

A basic plug-in must consist of at least two files.

1) The actual plug-in
2) The plug-in meta data (including version, author, update url etc)

The main plug-in file must be in the format: `([a-z0-9_\/]+).plugin.php`

The meta data file must be in the format: `([a-z0-9_\/]+).meta.plugin.php`

As of Bluetrait 2.0.0-alpha-2 it is now possible to store these files within a folder (for example "`bt-content/bt-plugins/awesome_stuff/awesome_stuff.plugin.php`"). The folder name does not need to be related to the plug-in name, but it is recommended.

The folder name must be in the format `([a-z0-9_\/]+)`

The Meta Data File

The meta data file provides information about your plug-in that Bluetrait can use.

The file must consist of a class and a single function. The class must be the start of the name of your plug-in file. So if your plug-in is called `awesome_stuff.plugin.php` the class must be called "`awesome_stuff`".

Inside the class must be a function called "`meta_data`" this function must return an array of values that Bluetrait can use. The array can have the following items:

```
$array = array (
      'plugin_name' => 'Example 1',
      'plugin_description' => 'An example plugin',
      'plugin_update_checker_url' => 'http://example.com/check.php',
      'plugin_author' => 'Author Name',
      'plugin_author_website' => 'http://www.example.com/',
      'plugin_website' => 'http://www.example.com/plugin/',
      'plugin_version' => '1.0'
);
```

You can see an example of this file in the code examples folder (`test.meta.plugin.php`).

The Plug-in File

The plug-in file is where the plug-ins logic is stored.

There is no specific layout required for this file but there is one main recommendation. This is that the plug-in should contain the following line of code before any other processing:

```
if (!defined('BT_ROOT')) exit;
```

This stops the file from being directly run from within a web browser. This can increase the security of the plug-in.

The next step is to hook into Bluetrait this is done through the following two functions:

- `bt_add_task($plugin_name, $section, $function, $priority = 10, $arguments = 1)`
- `bt_add_content_filter($plugin_name, $section, $function, $priority = 10, $arguments = 1)`

The first function (`bt_add_task`) allows the plug-in to register custom functions that Bluetrait will call when it is processing a certain section.

The second function (`bt_add_content_filter`) allows the plug-in to register custom functions that modify strings (for example modifying the post body)

The arguments for these functions are:

- `$plugin_name` - This is the name of your plug-in (for example "`awesome_stuff`")
- `$section` – This is the section where the hook should be added (for example "`common_loaded`")
- `$function` – This is the name of your custom function (for example "`awesome_stuff_init`")
- `$priority` – This is the priority of your function (1 being the highest).
- `$arguments` – This can be left blank as it not yet used.

You can see an example of this file in the code examples folder (`test.plugin.php`).

Let's get started.

So let's write a plug-in.

The first plug-in will be very basic and will simply display "Hello World" at the top of your website.

The first step is to decide on a plug-in name. This is an import step as the plug-in name will be used in multiple sections.

It is recommended that the plug-in name be unique as you cannot have two plug-ins with the same name.

Let's call our plug-in "Example 1". Therefore the file structure will look like this:

- `example_1.plugin.php` (the plug-in file)
- `example_1.meta.plugin.php` (the plug-in meta file)

Now we need to create the meta file. The meta file will look like this:

```
<?php

class example_1 {

	function meta_data() {

		$array = array (

		'plugin_name' => 'Example 1',

		'plugin_description' => 'An example plugin',

		'plugin_update_checker_url' => '',

		'plugin_author' => 'Author Name',

		'plugin_author_website' => 'http://www.bluetrait.org/',

		'plugin_website' => 'http://www.bluetrait.org/',

		'plugin_version' => '1.0'

		);

		return $array;

	}

}

?>
```

As can be seen here the class name is "`example_1`" the same as the start of the plug-in file name. The array must contain all the items even if they are empty (for example `plugin_update_checker_url`).

Now we can start on the actual plug-in.

```
<?php

if (!defined('BT_ROOT')) exit;


bt_add_task('example_1', 'common_loaded', 'example_1_function');


function example_1_function() {

      echo 'Hello World';

}

?>
```

So the first section stops the plug-in from being run directly from the web browser.

The second section adds a task. The first argument is the plug-in name; the second is the section where the plug-in hooks into Bluetrait and the third is the custom function.

The last section is the actual function that outputs “Hello World”. Please note the function name “`example_1_function`”. Prefixing the function name with the name of the plug-in helps to keep the function unique as you cannot have two functions with the same name.

The most important section of this example plug-in is probably the second argument for `bt_add_task`, in this case “`common_loaded`”.

Throughout Bluetrait there are sections or hooks. These sections are processed at a certain time. In the case of “`common_loaded`” this occurs at the end of the processing in `bt-common.php`. You may notice that the plug-in message is also displayed in the admin panel. This is because `bt-common.php` is used throughout Bluetrait.

If you only wanted the message to be displayed on the website (not admin panel) then you could use the section/hook “`header_loaded`”.

These sections/hooks are outlined later in this document.

Using the database

Once you've got your first plug-in working you'll probably be interested in storing some data somewhere.

Bluetrait uses a database for storing almost all data. For example: posts, comments, users, events, permissions and more.

When working directly with the database (and not through Bluetrait functions) you must be careful not to overwrite or delete important information. It is also not recommended that you modify the database structure as this can cause issues (such as when updating to a newer version of Bluetrait).

If your plug-in needs to store data that doesn't fit into the database structure it is best to create your own table for your database.

Bluetrait will not touch any table that wasn't directly created by it.

Developers should also be aware that the underlying database system may be different for each user. By default Bluetrait 2 works with MySQL and SQLite but it is possible that future versions of Bluetrait will work with more database systems (such as PostgreSQL). Therefore care should be taken when writing SQL statements as the syntax may be different for each database system.

If you are unable to support a certain database system please make this clear to the end user.

How it works

Bluetrait 2 uses the PHP Data Objects (PDO) class to support multiple databases. The database connection can be access through the `$bt_db` variable, while the table names can be accessed through the `$bt_tb` variable.

Here is an example:

```
$query = "SELECT * FROM $bt_tb->users WHERE user_id = ? LIMIT 1";

$stmt = $bt_db->prepare($query);

$stmt->bindParam(1, $user_id);

try {

      $stmt->execute();

}

catch (Exception $e) {

      bt_die($e->getMessage());

}

$user_details = $stmt->fetch(PDO::FETCH_ASSOC);
```

If you are trying to access the database from within a function you must make sure that `$bt_db` and `$bt_tb` are defined as `globals`.

Here is an example:

```
function example() {

      global $bt_db, $bt_tb;

      //database query here

}
```

To learn more about the PHP PDO syntax please read the documentation available at the PHP website (http://www.php.net/pdo).

A list of the table names can be found later in this document.

Storing data in the database

If your plug-in only needs to store a small amount of data in the database then the best method for this is to use the built in Bluetrait functions; otherwise it may be best to create your own table (this is explained in the next section).

First create the value within the database:

```
bt_add_config('awesome_stuff_config', 'my config');
```

You can then read and update the configuration value.

To read: `bt_get_config('awesome_stuff_config');`

To update: `bt_set_config('awesome_stuff_config', 'new config');`

To delete: `bt_delete_config('awesome_stuff_config');`

It is possible to store an array within this system. Example:

```
bt_add_config('awesome_stuff_config', $array);

$array = bt_get_config('awesome_stuff_config');
```

Handling multiple database systems

Sometimes there are syntax differences between database systems. The following example shows how to overcome this issue.

```
function bt_is_installed() {

	global $bt_db, $bt_tb, $bt_db_type;


	switch ($bt_db_type) {

		case 'mysql':

		$stmt = $bt_db->prepare("SHOW TABLES LIKE '$bt_tb-
	>site'");

		break;


		case 'sqlite':

		$stmt = $bt_db->prepare("SELECT * FROM SQLITE_MASTER
	WHERE tbl_name='$bt_tb->site'");

		break;

	}

	try {

		$stmt->execute();

	}

	catch (Exception $e) {

		bt_die($e->getMessage());

	}

	$array = $stmt->fetchAll(PDO::FETCH_ASSOC);

	if (!isset($array[0])) return false;

	return true;

}
```

The `$bt_db_type` variable contains the currently in use database type. The switch statement allows different queries to be run depending on the database type. `$bt_db_type` must also be set to `global` if used within a function. Currently MySQL and SQLite are supported.

Creating your own database tables

The following code (next page) shows how to create a database table within Bluetrait 2.

The `$bt_tb_prefix` variable contains the currently in use table prefix. This can be useful when creating a new database table. It is recommended that you always prefix your database table name with the Bluetrait prefix.

If you want to use the `$bt_tb` variable for your table name you can use the following code:

```
$bt_tb->add_table("table_name_without_prefix");
```

You can then use `$bt_tb->table_name_without_prefix` within your code.

Please note that the `add_table` command is not permanent so this line of code would need to be run each time when the plug-in is loaded.

If the end user is using MySQL Bluetrait will automatically optimise the table monthly, but only if the `add_table` function is used.

Creating Database Table Example:

```
switch ($bt_db_type) {
      case 'mysql':
            $query = "CREATE TABLE {$bt_tb_prefix}site (
              config_name varchar(255) NOT NULL,
              config_value TEXT NOT NULL,
              PRIMARY KEY  (config_name)
            ) DEFAULT CHARSET=utf8;
            ";
            try {
                  $bt_db->exec($query);
            } catch (Exception $e) {
                  bt_die($e->getMessage());
            }
      break;
      case 'sqlite':
            $query = "CREATE TABLE {$bt_tb_prefix}site (
              config_name varchar(255) NOT NULL,
              config_value varchar(255) NOT NULL,
              PRIMARY KEY  (config_name)
            );
            ";
            try {
                  $bt_db->exec($query);
            } catch (Exception $e) {
                  bt_die($e->getMessage());
            }
      break;
}
```

Checking for a database table

You may want to check if a table exists before trying to create it. The following code is an example of how to check for a database table:

```
switch ($bt_db_type) {
      case 'mysql':
      $stmt = $bt_db->prepare("SHOW TABLES LIKE '$bt_tb->site'");
      break;

      case 'sqlite':
      $stmt = $bt_db->prepare("SELECT * FROM SQLITE_MASTER WHERE tbl_name='$bt_tb->site'");
      break;
}
try {
      $stmt->execute();
} catch (Exception $e) {
      bt_die($e->getMessage());
}
$array = $stmt->fetchAll(PDO::FETCH_ASSOC);
if (!isset($array[0])) {
      //table doesn't exist
}
else {
      //table exists
}
```

Plug-in Case Study: The Contact Form

The contact form plug-in is the default plug-in included with Bluetrait 2. It adds a contact form function.

This plug-in makes use of “custom content”.

Custom content is a way of adding completely new functionality to the main website handled by Bluetrait.

In this case the plug-in adds a contact form but other functionality is possible, such as a photo galleries, atom feeds and more.

By default Bluetrait 2 has the following built in “content types”:

- Blog
- CMS
- RSS (for posts and comments)

When a user visits a Bluetrait powered website Bluetrait decides the content type that is currently in use. By default the blog type is used.

If no content type can be found Bluetrait allows a plug-in to capture the processing, allowing it to add new functionality.

Plug-in Security

Plug-ins can open Bluetrait up to security issues as there is no way of restricting what a plug-in can do.

Please take care when writing plug-ins.

SQL injection

A common form of attack is an SQL injection. As Bluetrait uses the PHP PDO class it is possible to use "Prepared statements".

Prepared statements are more secure as the database is able to differentiate between the actual SQL statement and user input.

An **insecure** SQL query may look like this:

```
SELECT * FROM users WHERE user_name = $_POST['user_name']
```

A more secure way of writing this is with prepared statements:

```
SELECT * FROM users WHERE user_name = ?
```

You then bind `$_POST['user_name']` to `?`

A full example:

```
$query = "SELECT * FROM users WHERE user_name = ?";

$stmt = $bt_db->prepare($query);

$stmt->bindParam(1, $_POST['user_name']);

$stmt->execute();
```

Cross-site scripting (XSS)

Another common attack is cross-site scripting (XSS). This is where a user is able to insert malicious code into the website. This code is then displayed to other users.

These attacks can steal cookies; infect the end users computer with spyware and more.

It is important that any user input is validated.

The most basic method is converting everything into HTML entities.

An **insecure** plug-in may include the following code:

```
<?php echo $user_input; ?>
```

A more secure way is to use Bluetrait's built in HTML entities function:

```
<?php echo bt_htmlentities($user_input); ?>
```

Plug-in Updating

Bluetrait 2 includes a plug-in update notification system. This system notifies the end user when a newer version of a plug-in is available.

The notification system uses Bluetrait's cron system to retrieve plug-in information daily.

To activate this service for your plug-in you must specify the plug-in update URL in the plug-in meta data file.

For example:

```
'plugin_update_checker_url' =>
'http://update.bluetrait.org/bluetrait2/update_check.php'
```

This file must be hosted by the developer and may be either static or dynamic.

Bluetrait expects the URL to return an XML document in the following format:

```
<?xml version="1.0" ?>

<plugin_name>

      <version>Version Number</version>

      <release_notes>Release Notes URL</release_notes>

      <download>Download URL</download>

</plugin_name>
```

Developers are free to use their own notification system if preferred.

Section Hooks (API)

Hook	Location	Description	Data	Version Introduced
admin_text_editor	Multiple	Used to display text editor in admin panel.	Passes an array with textarea name and content	2.0.0
admin_header_loaded	BT_ADMIN . '/include/admin-header.php'	Runs at the end of admin-header.php	None	2.0.0
admin_home_side_bar	BT_ADMIN . '/index.php'	Runs at the bottom of the right hand side bar	None	2.0.0
admin_home_body	BT_ADMIN . '/index.php'	Runs in the body	None	2.0.0
plugins_loaded	bt-common.php	Runs once plug-ins are all loaded	None	2.0.0
common_loaded	bt-common.php	Runs at the end of bt-common.php	None	2.0.0
header_loaded	bt-header.php	Runs near the bottom of bt-header.php before themes are loaded	None	2.0.0
shutdown	All	Runs at the end of any processing	None	2.0.0
save_config	BT_INCLUDE . '/functions.php'	Runs every time bt_save_config is called.		2.0.0
set_config	BT_INCLUDE . '/functions.php'	Runs every time bt_set_config is called		2.0.0
posts	BT_INCLUDE . '/posts.class.php'	Runs whenever posts are retrieved from the db.	Passes an array of retrieved posts.	2.0.0
content_identifier_defaults	BT_INCLUDE . '/functions.php'	Runs before any URL data relating to the content is set	Passes content identifier array	2.0.0
content_identifier	BT_INCLUDE . '/functions.php'	Runs after URL data has been collected	Passes content identifier array	2.0.0
theme_type_{$variable}	bt-header.php	Used to start the theme	None	2.0.0

comment_form	BT_CONTENT . 'bt-themes/classic/blog/comment_form.php	Runs in comment html form	None	2.0.0

Content Filter Hooks (API)

Hook	Location	Description	Data	Version Introduced
post_title	BT_INCLUDE . '/posts-template.functions.php'	The post title is passed through this filer	post title	2.0.0
post_body	BT_INCLUDE . '/posts-template.functions.php'	The post body is passed through this filter	post body	2.0.0
comment_body	BT_INCLUDE . '/comments-template.functions.php'	The comment body is passed through this filter	comment body	2.0.0
content_title	BT_INCLUDE . '/content-template.functions.php'	The content title I passed through this filter	content title	2.0.0
content_body	BT_INCLUDE . '/content-template.functions.php'	The content body is passed through this filter	content body	2.0.0

Database Tables and Structure

The "Site" Table

The site table is where the majority of Bluetrait's settings are stored.

Configuration Name	Description	Data	Version Introduced
category_permalink_format		STRING	2.0.0
comments	Allow Comments	0 or 1	2.0.0
content_permalink_format		STRING	2.0.0
cookie_name		STRING	2.0.0
cron_intervals	Stores time intervals for when cron tasks need to run	Serialized Array	2.0.0
current_theme		STRING	2.0.0
database_version		INT	2.0.0
description	Site Description	STRING	2.0.0
domain	Site Domain Name	lower case domain name	2.0.0
https	Enable HTTPs	0 or 1	2.0.0
installed	Date Bluetrait was installed	DATETIME	2.0.0
last_update_response	Built in update checker	XML	2.0.0
limit_posts	Number of posts on the front page	INT	2.0.0
name	Site Name	STRING	2.0.0
next_cron_run	Time when cron should run	DATETIME	2.0.0
plugin_data	Plug-ins to be loaded	ARRAY	2.0.0
plugin_update_data	Built in plug-in update checker	ARRAY	2.0.0
port_number	Port Number the site runs on (e.g 80)	INT	2.0.0
post_permalink_format		STRING	2.0.0
program_version		STRING	2.0.0
rss	Enable RSS feeds	0 or 1	2.0.0
script_path		STRING	2.0.0
themes	Enable/Disable Themes	0 or 1	2.0.0
time_zone		+/- INT	2.0.0

Other Useful Functions

Function Name	Description	Data	Return	Version Introduced
bt_check_date	Checks a date	Date (DD-MM-YYYY)	TRUE/FALSE	2.0.0
bt_check_datetime	Checks a datetime	DD-MM-YYYY HH:MM:SS	TRUE/FALSE	2.0.0
bt_remove_end_slash				2.0.0
bt_check_email_address				2.0.0
bt_get_config				2.0.0
bt_add_config				2.0.0
bt_set_config				2.0.0
bt_hard_set_config				2.0.0
bt_save_config				2.0.0
bt_die				2.0.0
bt_stop				2.0.0
bt_htmlentities				2.0.0
bt_set_header				2.0.0
bt_send_headers				2.0.0
bt_datetime				2.0.0
bt_datetime_utc				2.0.0
bt_datetime_utc_from_datetime				2.0.0
bt_ip_address				2.0.0
bt_x_title				2.0.0
bt_add_magic_quotes				2.0.0
bt_remove_magic_quotes				2.0.0
bt_get_user_data				2.0.0
bt_set_user_data				2.0.0
bt_is_logged_in				2.0.0
bt_uuid				

Cron

Bluetrait 2 includes a basic cron system. This system is designed to process background tasks at a set interval. This can be useful for tasks that are required to run without any user intervention such as publishing future posts.

Bluetrait has four built in time intervals:

- Every Hour
- Every Day
- Every Week
- Every Month

The cron system is setup so that it will only run at most once every hour (to keep server load down).

To add a task to the cron system is it simply a matter of adding a task within your plug-in.

The cron hooks are as follows:

- `cron_every_hour`
- `cron_every_day`
- `cron_every_week`
- `cron_every_month`

So an example of adding a task to run every hour:

```
bt_add_task('awesome_stuff', 'cron_every_hour', '
awesome_stuff_hourly_process');
```

Cron tasks are not permanent so this line of code would need to be run each time when the plug-in is loaded.

The cron system is triggered by any request to a page handled by Bluetrait (for example: front page, rss etc), therefore the cron system may not run exactly every hour. Sites with higher visits are more likely to run closer to the schedule.

Conclusion

Hopefully this guide has provided enough information about Bluetrait to allow anyone to create a plug-in.

Thank you for developing for Bluetrait; your time and dedication is what makes a good product great.

Again if you require help with any aspect of Bluetrait please contact me:

Michael Dale
mdale@dalegroup.net