Pluma Coding Standards

Pluma follows the [PSR-2](http://www.php-fig.org/psr/psr-2/) coding standard and [PSR-4](http://www.php-fig.org/psr/psr-4/) autoloading standard.

In addition, the content of this document should be followed by core developers, extension package authors, and code contributors when writing for and/or extending Pluma.

The Pluma Coding Standard apply to written code within Pluma and its contributed extension packages and modules.

(Some parts of Pluma code might not adhere to the standard, but core developers are working gradually to improving compliance).

The Coding Standards are version-independent and may be subject to change in every major version. All new code should follow the current standards, regardless of Pluma version.

### Basic Naming Convention

This section of the standard comprises what should be considered the standard coding elements that are required to ensure a high level of technical interoperability between shared code.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119](http://www.ietf.org/rfc/rfc2119.txt).

# Definition of Terms

The Coding Standards are version-independent and may be subject to change in every major version. All new code should follow the current standards, regardless of Pluma version.

“Core Code”, or “Infrastructure Code” refers to code that make up the Pluma Framework which serves as the backbone for the application. Core Code files are all located inside the folder *core* (except files inside *core/submodules*). Files inside the *core/submodules* folder are collectively referred to as “Base Modules”, and is the code for the default content management system (Pluma CMS).

“Module” refers to a folder containing a collection of files serving a dedicated purpose within the application. Files can be controllers, models, views, assets, and helper functions. Examples include “*User*” module, dedicated to managing users of the app, (among other things); and “*Role*” module, a submodule and a Role Based Access Control (RBAC) library for *User*.

“Submodule” refers to a module inside a module. Submodules are located inside the folder named *submodules* within it’s parent module. A submodule can also have its own submodules. There is no hard limit on the depth of submodules, but it is RECOMMENDED to stop at the third level.

“Presentation Layer”, or “Visual Layer”, refers to code *presenting* the data to the user. The files included are *markup templates* (HTML, PHP), and *stylesheets* (CSS).

“Functional Layer” refers to any javascript code within the presentation layer.

“Frontend Layer” refers to the collection of presentation and functional layer.

“Logical Layer” refers to any code manipulating, evaluating, and/or computing (but not presenting) data, and is written in PHP.

“Database Layer” refers to database schema.

“Backend Layer” refers to the collection of logical and database layer.

### Code Demarcation Standard

Pluma core developers MUST follow the Pluma code demarcation standard.

This standard is RECOMMENDED for third-party extension authors.

The standard was developed in the scope of efforts to ensure the following:

1. Decouple the presentation layer from the functional layer.
2. Decouple the functional layer from the logical layer.
3. Reestablish emphasis on decoupling HTML, CSS, and JS from PHP classes.

# Semantic