XOOPS 2.6.0 Goes PSR-4 plus Other Core Enhancements - XOOPS

NEWS_PDF_AUTHOR: Mamba

NEWS_PDF_DATE: 2015/10/24 8:10:00

1) **Richard** is powering forward with some very cool enhancements for the Core of XOOPS 2.6.0. Recently he committed **these enhancements**: Quote:

> First stage of localization refresh #261, and other misc cleanup - Implements localized date and time handling. PHP DateTime objects used extensively in all date processing. Localization is provided by [Punic](https://github.com/punic/punic) - Add `Request::getDateTime()` Returns a `\DateTime` object from `Form\DateSelect` and `Form\DateTime` input. The form classes can also receive DateTime objects as values. Form dates are now local to the user, or system default if no user. - Timezones are now PHP `DateTimeZone` names, not float offsets. PHP DateTime and DateTimeZone objects are used to implement all calculations. This makes handling of things like daylight savings or summer time automatic. - Add `Dtype::TYPE_TIMEZONE` to store and restore \DateTimeZone objects - Add Smarty `datetime` modifier, allowing both unix timestamps and DateTime objects to be processed by XoopsLocale::formatTimestamp(), allowing date and time formatting to be controlled in the presentation layer with tags like `` - XoopsList class broken into individual classes in `Core\Lists`. These class are expected to implement `Core\Lists\AbstractList`. These can interact directly with form fields, reducing code complexity and duplication. - Add select_editor 'formtype' for $modversion['config']. This removes the need for active code to load editors in a module's xoops_version.php. All editor lists now originate from one point, Core\Lists\Editor - New subclass of Form\Element, OptionsElement which provides the standard option methods. List classes can interact with any extender of OptionsElement, see Core\Lists\AbstractList::setOptionsArray() - `Xoops_*` classes moved to actual `Xoops` namespace. These *PSR-0*, pseudo namespaces are obsolete. Their removal allows us to move to PSR-4 based loading for the Xoops namespace. - CountryFlag service no longer delivers bare URL to resource, only HTML to display flag. This allows more flexibility on how the flag is presented. - Accept `Trowable` in `Core\Logger::handleExeception()` (PHP7)

He also posted a **RFC (Request for Comments)** on three issues: - Namespaces - Moduels vs. Extensions - Add a modinfo column to system_module Quote:

> I've given a lot of consideration to modules, and have a few ideas I want to present. If you have comments, alternatives, objections or other feedback, please respond. Baring objections, I would like to move on this quickly. **Namespaces** Add a namespace column

to system_module table. This would be the PHP namespace for the module's code. It would map to the module's class directory in PSR-4 fashion. This would be implemented automatically for all active modules with a namespace specified. Note, this will be separate from the composer maintained loader, since composer has no way of knowing if a module is "active" in XOOPS. Autoloading components from modules which are not installed or inactive could create lots of issues. With the namespace enabled, most of the old directory positional naming constructs can be replaced with objects. For example, instead of using the $modversion['onInstall'] entry to locate a file to include, then building function names based on the module name, the code would look more like this:

```
 if (class_exists($moduleNamespace . 'SystemInstall')) {
    $install = new $moduleNamespace . 'SystemInstall;
    $success = $install->postInstall($module);
}
```

Our current preloads directory, with possibly multiple files, would disappear, and everything would reside in the$moduleNamespace . 'System\Events' class instead. In this scheme, the class/System directory would be reserved for the system specified classes. Each would have a specific interface defined. There are more details, but this should illustrate the concept. **Modules vs. Extensions** Based on some thoughts from @bitcero and some more consideration, I propose eliminating the extension concept as is, and moving to a module category system to make management cleaner. Categories would be something like this: Category: Usage ---------- **Content:** articles, blogs, calendars, forums, etc. **Developer:** module builders, schema tools **Extension:** service providers, system enhancement **Locale:** i18n resources, language packs **Theme:** for the new module based themes This is not carved in stone, but the idea is to create a limited, high level taxonomy to categorize modules. The defined categories would be enforced, as we don't need a free for all. The module listing functions would be adapted to deliver lists of individual categories. A category column on the system_module table would hold this item. To support subtle differences, each category would be backed by a sub class of the system module class, but all would offer the same functionalities. **Add a modinfo column to system_module** Caching of the $modversion array has been suggested many times. Adding a single column set on install and update can accomplish this, and the existing getInfo() and related methods will be able to use the column data as the source, eliminating the need to hit the filesystem. With this information in the database, all the module object access can be fully cached, which should speed thing up a bit.

Please respond directly on GitHub It is also worth to mention that **Eduardo** is making some major progress on creating a new "Presentation Layer" in XOOPS 2.6.0 based on his excellent "Common Utilities" **Slowly but surely, the pieces of the puzzle called XOOPS 2.6.0 are starting to fall in place!** 

1) **Richard** is powering forward with some very cool enhancements for the Core of XOOPS 2.6.0. Recently he committed **these enhancements**: Quote:

> First stage of localization refresh #261, and other misc cleanup - Implements localized date and time handling. PHP DateTime objects used extensively in all date processing. Localization is provided by [Punic](https://github.com/punic/punic) - Add `Request::getDateTime()` Returns a `\DateTime` object from `Form\DateSelect` and `Form\DateTime` input. The form classes can also receive DateTime objects as values. Form dates are now local to the user, or system default if no user. - Timezones are now PHP `DateTimeZone` names, not float offsets. PHP DateTime and DateTimeZone objects are used to implement all calculations. This makes handling of things like daylight savings or summer time automatic. - Add `Dtype::TYPE_TIMEZONE` to store and restore \DateTimeZone objects - Add Smarty `datetime` modifier, allowing both unix timestamps and DateTime objects to be processed by XoopsLocale::formatTimestamp(), allowing date and time formatting to be controlled in the presentation layer with tags like `` - XoopsList class broken into individual classes in `Core\Lists`. These class are expected to implement `Core\Lists\AbstractList`. These can interact directly with form fields, reducing code complexity and duplication. - Add select_editor 'formtype' for $modversion['config']. This removes the need for active code to load editors in a module's xoops_version.php. All editor lists now originate from one point, Core\Lists\Editor - New subclass of Form\Element, OptionsElement which provides the standard option methods. List classes can interact with any extender of OptionsElement, see Core\Lists\AbstractList::setOptionsArray() - `Xoops_*` classes moved to actual `Xoops` namespace. These *PSR-0*, pseudo namespaces are obsolete. Their removal allows us to move to PSR-4 based loading for the Xoops namespace. - CountryFlag service no longer delivers bare URL to resource, only HTML to display flag. This allows more flexibility on how the flag is presented. - Accept `Trowable` in `Core\Logger::handleExeception()` (PHP7)

He also posted a **RFC (Request for Comments)** on three issues: - Namespaces - Moduels vs. Extensions - Add a modinfo column to system_module Quote:

> I've given a lot of consideration to modules, and have a few ideas I want to present. If you have comments, alternatives, objections or other feedback, please respond. Baring objections, I would like to move on this quickly. **Namespaces** Add a namespace column to system_module table. This would be the PHP namespace for the module's code. It would map to the module's class directory in PSR-4 fashion. This would be implemented automatically for all active modules with a namespace specified. Note, this will be separate from the composer maintained loader, since composer has no way of knowing if a module is "active" in XOOPS. Autoloading components from modules which are not installed or inactive could create lots of issues. With the namespace enabled, most of the old directory positional naming constructs can be replaced with objects. For example, instead of using the $modversion['onInstall'] entry to locate a file to include,

then building function names based on the module name, the code would look more like this:

```
 if (class_exists($moduleNamespace . 'SystemInstall')) {
    $install = new $moduleNamespace . 'SystemInstall;
    $success = $install->postInstall($module);
}
```

Our current preloads directory, with possibly multiple files, would disappear, and everything would reside in the$moduleNamespace . 'System\Events' class instead. In this scheme, the class/System directory would be reserved for the system specified classes. Each would have a specific interface defined. There are more details, but this should illustrate the concept. **Modules vs. Extensions** Based on some thoughts from @bitcero and some more consideration, I propose eliminating the extension concept as is, and moving to a module category system to make management cleaner. Categories would be something like this: Category: Usage ---------- **Content:** articles, blogs, calendars, forums, etc. **Developer:** module builders, schema tools **Extension:** service providers, system enhancement **Locale:** i18n resources, language packs **Theme:** for the new module based themes This is not carved in stone, but the idea is to create a limited, high level taxonomy to categorize modules. The defined categories would be enforced, as we don't need a free for all. The module listing functions would be adapted to deliver lists of individual categories. A category column on the system_module table would hold this item. To support subtle differences, each category would be backed by a sub class of the system module class, but all would offer the same functionalities. **Add a modinfo column to system_module** Caching of the $modversion array has been suggested many times. Adding a single column set on install and update can accomplish this, and the existing getInfo() and related methods will be able to use the column data as the source, eliminating the need to hit the filesystem. With this information in the database, all the module object access can be fully cached, which should speed thing up a bit.

Please respond directly on GitHub It is also worth to mention that **Eduardo** is making some major progress on creating a new "Presentation Layer" in XOOPS 2.6.0 based on his excellent "Common Utilities" **Slowly but surely, the pieces of the puzzle called XOOPS 2.6.0 are starting to fall in place!**