Sunrise Architecture Review

March 6th, 2014

- https://github.com/newclarity/sunrise

This architecture review is a study to spark discuss of architecture options for the WordPress Metadata Feature-as-Plugin *(WPMeta for this document.)*

The result of 4+ years of focused R&D and many painful lessons learned.

Outline

21. Sunrise::the_field()
22. Sunrise::field() Filters
23. Storage
24. Forms: Object Type and Form Context
25. Main Form and Metaboxes
26. Object Factories and Reuse
27. Error Handling
28. Form/Field Validation
29. Sanitization
30. Hooks and Autoloader
31. Supports Use/Extension at 3 Expertise Levels

1. QuickStart

Ideally a Must-used Plugin

- Needs a plugin loader:

```
<?php
/**
 * Plugin Name: {Your Site}'s Must-Use Plugins
 * Description: Contains the WordPress/PHP plugins for {Your
Site}.
 */
require(__DIR__ . '/sunrise/sunrise.php');
```

Registering Forms and Fields

- Just use `'init'` hook.
- Example:

```
<?php
/**
 * functions.php - The functions file for {Your Site}'s theme.
 */
add_action( 'init', 'yoursite_init' );
```

```
function yoursite_init() {

  Sunrise::register_form( 'your_post_type' );

  Sunrise::register_form_field( 'website', array(
    'type'  => 'url',
    'label' => __( 'Website', 'your-domain' ),
    'html_placeholder' => 'http://www.example.com',
    'html_size'  => 50,
  ));

  Sunrise::register_form_field( 'tagline', array(
    'label' => __( 'Tagline', 'your-domain' ),
    'html_size'  => 50,
  ));

  Sunrise::register_form_field( 'blurb', array(
    'type'  => 'textarea',
    'label' => __( 'Blurb', 'your-domain' ),
    ));
}
```

2. The Sunrise Class Drives the API

Four of the main static methods:

```
Sunrise::register_form( $object_type, $form_args )
Sunrise::register_field( $field_name, $field_args )
Sunrise::register_form_field( $field_name, $field_args, $multiuse )
Sunrise::add_form_field( $field_name )
```

"Helper" classes that implement the API:

```
_Sunrise_Forms_Helper
_Sunrise_Fields_Helper
_Sunrise_Html_Elements_Helper
_Sunrise_Post_Admin_Forms_Helper
_Sunrise_Posts_Helper
```

Not an important consideration for WPMeta since WordPress won't need a

namespacing mechanism.

3. Sunrise_Base Helper Methods

- **Static Hooks:**

```
self::add_static_action( $action, $method_or_priority = false,
$priority = 10 )
self::add_static_filter( $filter, $method_or_priority = false,
$priority = 10 )
```

  - Requires `get_called_class()`
    - Thus requires PHP 5.3;
    - So also **NOT** relevant for WordPress core.
  - Example:

```
class _Sunrise_Example {
  static function on_load() {
    self::add_static_action( 'init', 0 );
  }
  static function _init() {
    ...
  }
}
```

    *Sunrise*Example::on_load();

- **Instance Hooks:**

```
$this->add_action( $action, $method_or_priority = false, $priority
= 10 )
$this->add_filter( $filter, $method_or_priority = false, $priority
= 10 )
```

- **Scoped to instance**
- Extensibility for class, but as scope-limited it more robust than normal hooks.
- Planned *(Not yet implemented)*
  - But previously implemented on other projects

4. More Object-Oriented than Functional Style

- Required to manage complexity
- Not purist OOP, but
- All code in classes,
- Entities (Forms, Fields, etc.) are Object Instances
- Global Base Class and Base Classes for Major Objects

5. Forms, Fields, Form Types and Field Types

Instances:

- **Forms** - Collections of Fields. Not necessarily 1-to-1 for a `<form>`.
- **Fields** - Models a data entry field; `<input>`, `<select>`, `<textarea>`, etc.

Classes:

- **Form Types** - Class extending `Sunrise_Form_Base` or a *"Prototype" (an array of `$form_args`)*
- **Field Types** - Class extending `Sunrise_Field_Base` or a *"Prototype" (an array of `$form_args`)*

6. "Prototypes"

```
register_field_type( 'headshot', array(
  'type' => 'image',
  'aspect_ratio' => '3:4'
));
...
register_form( 'myapp_contact' );
register_form_field( 'photo', 'type=headshot' );
```

- Is concept of Sunrise
- Is a named array of object properties
- Javascript Prototypes are its namesake
- Works like CSS cascading of property values
- Not fully implemented yet

7. Lightweight Registration/Delayed Fixup & Instantiation

"Fixup" happens on `wp_loaded` instead of at registration.

Field instantiations are delayed until needed by a form.

This solves two (2) problems:

1. **Order of Form and Field Registration**
2. **Slower Performance for Larger Apps**

8. The Features Class for Fields

- Features are collections of HTML elements
- Fields are comprised of up to five (5) Features.
  - Control; `<input>`, `<select>`, `<textarea>`, etc.
  - Label - `<label>`
  - Help - `<div>` for persistent help text
  - InfoBox - `<div>` for on-hover/on-click help text
  - Message - `<div>` for warning/error text.

More Features could easily be added in future for other needs.

9. HTML Element Class

```
new Sunrise_Html_Element( $tag_name, $attributes = array(), $value = null, $reuse = false )
```

- Vs. hard-coded, allows property `$args` delegated down from `register_*()`

to HTML element.

- Used internally by Forms/Fields/Features/etc.
- Gives rise to Qualified and Unqualified `$args`.

10. The $args Parameter Pattern

```
function register_something( $name, $args = array() ) {
  $args = wp_parse_args( $args, array(
    'one_property'     => 'default value #1',
    'another_property' => 'default value #2',
  ));
  self::$_somethings[$name] = $args;
}
```

- Used for registration and other methods needing future flexibility.
- Allows future extension without method signature breakage.
- Critical to layered architecture of:
  - Form
  - Field
  - Feature
  - HTML Element
- `$args` can be passed as arrays or as URL-encoded strings (`foo=1&bar=2`)

11. Qualified and Unqualified Property $args

- **Naming Conventions**
  - Single word properties are object-prefix qualified:
    - `$form_name`, `$form_title`, `$form_hidden`, etc.
    - `$field_name`, `$field_type`, `$field_default`, etc.
    - `$html_name`, `$html_placeholder`, `$html_rows`, etc.
  - Multi word properties not object-prefix qualified:
    - Forms: `$object_id`, `$_object_type`, etc.
    - Fields: `$no_lablel`, `$*html*element, etc.

    - `NO_PREFIX` Fields:
        - Fields: `$value`, `$features`
        - Set with `NO_PREFIX` class constant
- Enables passing of contained class `$args`

```
Sunrise::register_form_field( 'website', array(
  'type'  => 'url',
  'label' => __( 'Website', 'your-domain' ),
  'html_placeholder' => 'http://www.example.com',
  'html_size'  => 50,
));
```

    - `VAR_ALIASES` planned
        - Shortcuts for unambiguous common properties
        - Example:

```
Sunrise::register_form_field( 'website', array(
  'type'  => 'url',
  'label' => __( 'Website', 'your-domain' ),
  'placeholder' => 'http://www.example.com',
  'size'  => 50,
));
```

12. Private, Protected and *"Internal"* Methods & Properties

- All properties and methods have a leading underscore that are:
    - **Private**
    - **Protected**
    - **Internal**
        - Used as if Private/Protected but must be public for scoping/callback
        - Actions and Filters methods are an example

13. Actions and Filters

- **Naming Conventions**
  - Where possible method names for hooks follow this pattern:
    - `"_{$method_name}"` - For priority `10`
    - `"_{$method_name}_{$priority}"` - For all other priorities
    - Examples: `function _init_0() { ... }`
  - Where not possible, such as for:
    - Shared hooks, or
    - Hook name is invalid method name syntax
    - Uses custom name:
      - `"_{$custom_name}"` - For priority `10`
      - `"_{$custom_name}_{$priority}"` - For all other priorities

14. CONST as Class Metadata

- Like Java annotations and .NET attributes.
- Currently *(will likely change, be added to.)*

Constant Name	Description
`VAR_PREFIX`	The prefix used for Qualified Names of Property `$args`.
`NO_PREFIX`	The pipe-separated list of field names not to prefix.
`CONTROL_TAG`	HTML <tag> name used for the main HTML element of the Control Feature.
`HTML_TYPE`	HTML @type attribute implemented for a Field class.
`FORM_CONTEXT`	Context of Form: `'admin'` or `'theme'`

```
class Sunrise_Url_Field extends Sunrise_Field_Base {
  const HTML_TYPE = 'url';
  ...
```

```
}
```

15. Object Type Classifiers

```
$classifier = new Sunrise_Object_Classifier( $classifier );
```

- Critical concept for Sunrise
- Allows single value to denote object type, i.e. `'post'`, `'user'`, `'comment'`
- Handles subtypes, i.e. `'post/page'`, `'post/myapp_contact'`
- Properties of object:
  - `$classifier->object_type`, i.e. `'post'`
  - `$classifier->subtype`, i.e. `'myapp_contact'`
- Object can be cast to a string, i.e. this prints 'Equals':

```
$classifier = new Sunrise_Object_Classifier( 'post/myapp_contact'
);
echo 'post/myapp_contact' == $classifier ? 'Equals' : 'Not
Equals';
```

16. Form and Field Identification:

- Forms are identified by:
  - Form Index
    - Runtime specific
    - Globally unique across Forms
  - Or unique combination of:
    - **Object Type**
    - **Form Context** - `'admin'` or `'theme'`
    - And **Form Name** - defaults to `'main'`
- Fields are identified by:
  - Field Index
    - Runtime specific
    - Globally unique across Fields

- Or unique combination of:
  - **Form**
  - And **Field Name**
- Or if Multiuse:
  - **Field Name**

17. Method Naming

- Class Methods naming:
  - `$this->thing()`
  - `$this->set_thing($new_thing)`
  - `$this->get_things()`
  - `$this->the_thing()`

18. Virtual Properties

- Virtual properties using `__get()`
- Accessing `$this->thing` will call `$this->thing()`, if exists
- Useful in building strings:
  - i.e. `"{$label->feature_html}{$control->feature_html}<div class=\"clear\"></div>"`

19. Template Tag Methods

- Template tag methods using `__call()`
- Accessing `$this->the_thing()` will run `echo $this->thing()`, if exists
- Makes for very consistent API interface

20. Sunrise::field()/field_html()

- Get the Field object:

```
$field = Sunrise::field( 'name', array( 'object_id' => $post->ID )
```

```
)
```

- Or output it's HTML

```
echo Sunrise::field_html( 'name', array( 'object_id' => $post->ID
) )
```

21. Sunrise::the_field()

- Planned
- These do the same:

```
echo Sunrise::field_html( 'name', array( 'object_id' => $post->ID
) )
Sunrise::the_field( 'name', array( 'object_id' => $post->ID ) )
```

- Or as an instance; know's it's own `$object_id`:

```
$field->field( 'name' )
```

22. Sunrise::field() Filters

- Planned
- Filters:
  - `'pre_get_field'`
  - `'get_field'`
  - `'empty_field'`
- Allows *"Virtual Fields"*
  - i.e. `$field->postal_address(), maybe comprised of:
    - `$field->street`
    - `$field->city`
    - `$field->region`
    - `$field->country`

        - `$field->post_code`
    - Like nested Russian dolls
- Sunrise-1 has this

23. Storage

- `wp_postmeta` by default
    - Currently stored as `"_sf[{$field_name}]"`
    - WPMeta should probably just be `"_{$field_name}"`
- Planned
    - Storage to `wp_term_relationship`
    - Storage to `wp_posts`
    - Storage to `wp_custom_table`
    - Storage anywhere *(images stored in Amazon S3?)*
- Subclassing or Containment?
    - Need input/discussion

24. Forms: Object Type and Form Context

- Currently using subclassing:
    - i.e. `Sunrise_Post_Admin_Form`
    - Need input/discussion

25. Main Form and Metaboxes

- For posts, The `'main'` form gets displayed between Title/URL and TinyMCE Content.
- Planned: Other forms get displayed in Metaboxes

26. Object Factories and Reuse

```
$form = Sunrise::create_form( $form_args );
$field = Sunrise::create_field( $field_args );
```

- Reuse is not fully baked yet
    - Maybe a 2nd `$reuse` parameter defaulting to no?

27. Error Handling

- Currently using `trigger_error()`
- Doesn't handle AJAX errors well yet.
- Need input on best approach.

28. Form/Field Validation

Envision CodeIgnitor-like validation:

```
register_form_field( 'email', array(
  ...
  'validation' => 'required|is_email|is_unique[users.email]',
);
```

- Pipe-separated list of registered keywords
- Site builders can register new keywords tied to a callable
- Have not previously implemented
- Need input on best approach

29. Sanitization

- Basic sanitization implemented.
- Intend more advanced sanitization on field types
- Intend to allow Field subclasses to override sanitization

30. Hooks and Autoloader

- No autoloader yet, but

- Helper classes add hooks required on page load,
- Instance classes only add instance-related hooks in `__construct()`, and thus
- Will allow instances to only be loaded when needed.

31. Supports Use/Extension at 3 Expertise Levels

1. **Themer** uses Forms/Fields in theme
2. **Site Builder** registers Forms/Fields/Prototypes/etc., adds instance hooks.
3. **Plugin Developer** builds new type classes: Forms/Fields/Features/etc.

END