



Greetings my name Mikel King

Let me start off by telling you a little about myself. I am currently the Lead Organizer for WordCamp NYC.

How many of you have seen me talk before whether it's in person or even on WordPress.tv?

Twitter:

@MikelKing

GitHub:

<https://github.com/mikelking>

LinkedIn:

<https://www.linkedin.com/in/mikelking>

x1:

mikelking

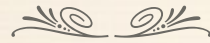
Contacting me is pretty straight forward:

1. It frames this as I am the authority on the subject and I believe that there are many different avenues
2. I don't feel like the quality answers are achievable in such a short period
3. I prefer intelligent conversations to volley ball Q & A

Those that have know that I really dislike the whole Q & A format and it not because I dislike answer questions it because...

Therefore, please save your questions and comments until the end. If I do not get to your question then come find me after this session in the help desk (a.k.a. Happiness Bar).

# Photo of me



*circa 1993 while in the USCG*



I work for a company called TMBI, until recently known as Readers Digest Association. When I refer to RD I am mostly focusing on rd.com. That's right I am from NY and I tend to talk fast. I will not be offended if you raise your hand and ask me to slow down just keep in mind that we have a lot of ground to cover and somethings me get bumped if I don't keep it moving.

I have been in technology for well over 20 years and served 11 years in the U.S. Coast Guard in fact this is a photo of me when I was in the military. As you can see a bit has changed most notably I no longer have the ZPM powered helmet.

## Aspects of this presentation are available on GitHub

- ❖ <https://github.com/mikelking/magickal-mu-plugins>
- ❖ [https://github.com/mikelking/singleton\\_base](https://github.com/mikelking/singleton_base)

# Some assumptions

- ❖ That the majority of you in this room are developers.
- ❖ That you have an understanding about what a MU plugin is.
- ❖ That you have at least dabbled in writing your own plugin.

In order to get the most out of MU plugins we need to cover a few things. First we need to make some assumptions.



- ❖ First how many of you actively code in PHP every day?
- ❖ How many practice OOP?
- ❖ How many have heard of codesniffing?
- ❖ How many have heard of unit testing?
- ❖ How many have heard of DRY?
- ❖ How many have heard of design patterns?
- ❖ How many people have sites that have hundreds even thousands of lines of code in their functions.php file?

By a show of hands please answer

Ask yourself Why do you do this? Why are you killing yourself? What do you do when you have to troubleshoot a bug that leads to the hundreds/thousands of lines of code in that one file?


Put that code in a plugin! Plugins are great because they are modular and you can activate and deactivate them to expedite troubleshooting.



Over 9000 lines of code

When I first arrived at RD 3 years ago hardly anything was pluginized. The functions.php have 9000 lines of code. The first I did was create a plugin framework that we still use today.





If it needs to be on the site  
then put it in a plugin!!!

Unfortunately there is a conundrum, because like the code in the functions.php MU plugins can not be activated or deactivated at will. The only way to deactivate a MU plugins is to physically remove it from the system. That's a major bummer and kind feels like a step backward.

# Problem solved MU Plugins

However, this plugin stub proved to be inefficient as we started to build more plugins. There was too much duplication of code from one plugin implementation to another. This lead me to develop a more efficient framework based on one of the core tenants of OOP inheritance and the DRY principle. For those of you who do not know about DRY it stands for "Don't Repeat Yourself". So I distilled the duplicative code into a library of functions and then contemplated how to make them consistently available to the entire project which of course lead naturally to MU plugins.

# No quite

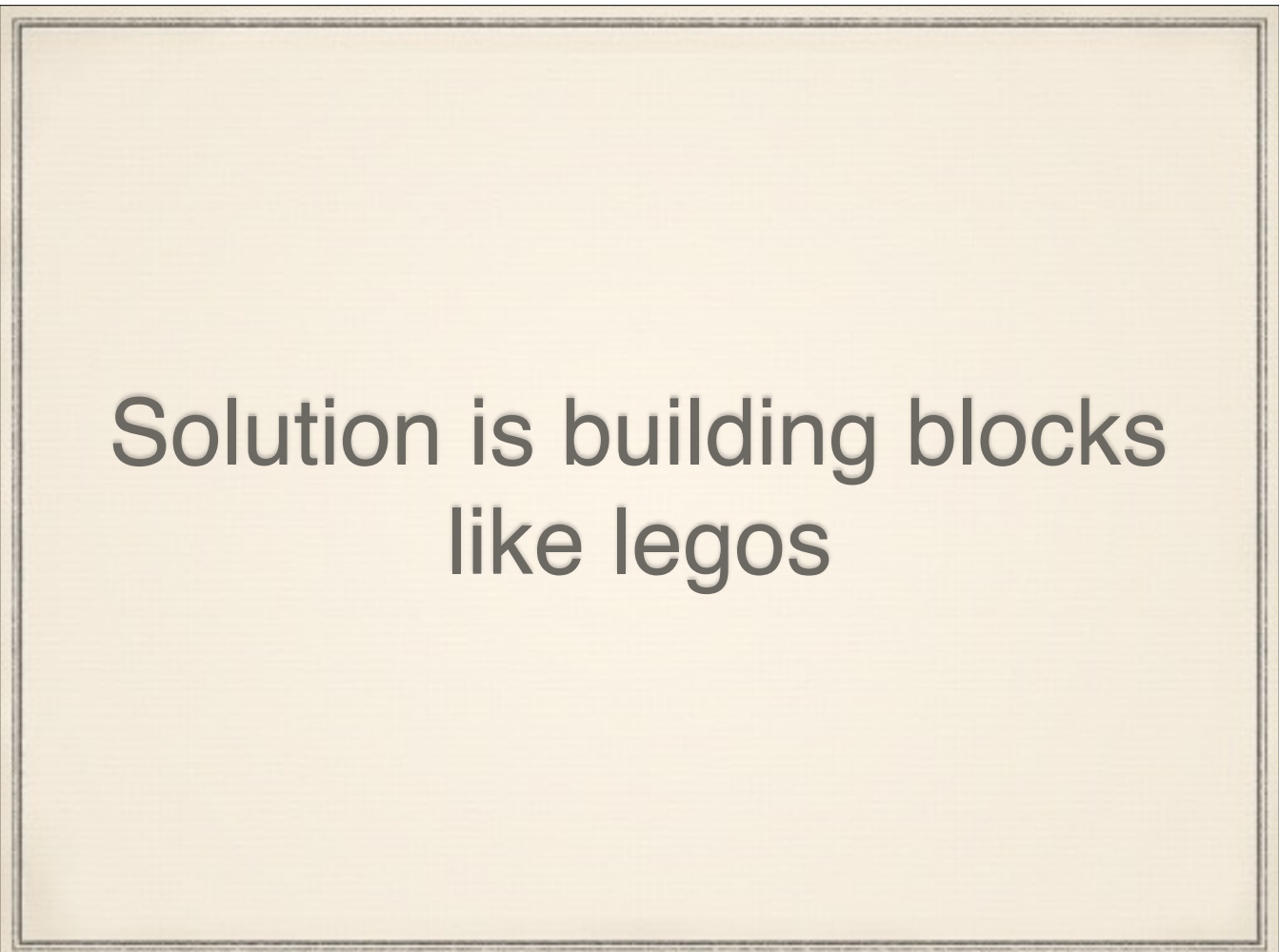
MU Plugins like the functions.php

- ❖ Can NOT be turned OFF
- ❖ Are active for ALL sites in a MultiSite
- ❖ Are pervasive across your entire WordPress Install

Unfortunately there is a conundrum, because like the code in the functions.php MU plugins can not be activated or deactivated at will. The only way to deactivate a MU plugins is to physically remove it from the system. That's a major bummer and kind feels like a step backward.

Feels like a huge bummer

You like someone sucked the jam out of your donut.



Solution is building blocks  
like legos

The solution was simple create building blocks of code 'like legos' that could be stacked together in a logical way within a standard plugin that can be activated and deactivated at the application layer. So this library of MU plugin code just sits there waiting until it is called upon and then springs into action. The library of code for this is hosted on my public GitHub.

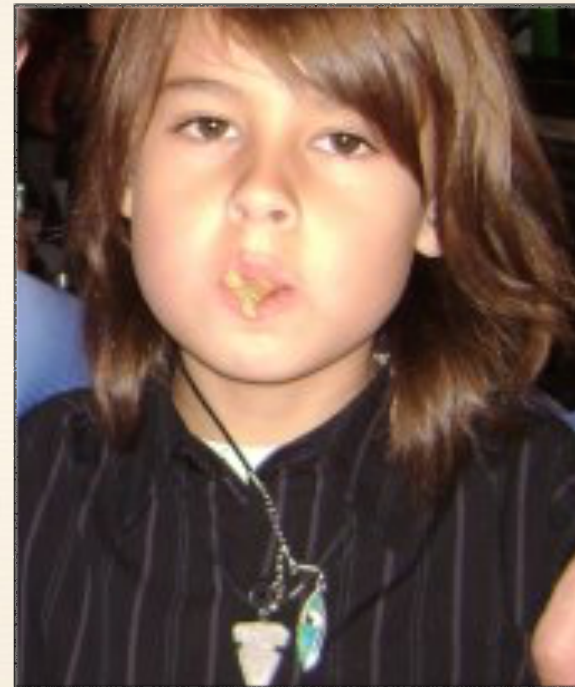
# This library of blocks

- ❖ Relies heavily on OOP
- ❖ Especially on the principles of inheritance

Everything in OOP begins with the class which is the fundamental building block of any OOP program. So before we delve too deep into that code I want to talk about the core facet of OOP as it pertains to MU plugins and that is inheritance. Without proper inheritance the mu plugin library would be untenable.



# Inheritance in people



Everything within the braces of a class is a property of that class. In people we would call these traits. So for the sake of this discussion we will take a brief look at some people.

# Inheritance in people

```
<?php
class Father extends Grand_Father {
    const FIRST_NAME = 'Mikel';
    const LAST_NAME  = 'King';
    const DEFAULT_TZ = 'America/New_York';
    const STATUS     = 'ALIVE';
    const EYE_COLOUR = 'Hazel';
    const ALLERGIES  = 'Chocolate';
}
```

```
<?php
class Son Extends Father {
    const FIRST_NAME = 'Jarett';
    const LAST_NAME  = 'King';
    const DEFAULT_TZ = 'America/New_York';
    const STATUS     = 'ALIVE';
}
```

# Execution

```
$mikel = Father::get_instance();  
$mikel->print_stats();  
print( '====' . PHP_EOL );  
  
$jk = Son::get_instance();  
$jk->print_stats();  
print( '====' . PHP_EOL );
```

```
Class name: Father  
Name: Mikel King  
Hair colour: Dark Brown  
Eye colour: Hazel  
Status: ALIVE  
Known allergies: Chocolate
```

```
Class name: Son  
Name: Jarett King  
Hair colour: Dark Brown  
Eye colour: Hazel  
Status: ALIVE  
Known allergies: Chocolate
```

# Inheritance in people



# Inheritance in people

```
<?php
class Grand_Father extends Great_Grand_Mother {
    const FIRST_NAME = 'James';
    const LAST_NAME  = 'King';
    const DEFAULT_TZ = 'America/New_York';
    const STATUS     = 'ALIVE';
}
```

```
<?php
class Father extends Grand_Father {
    const FIRST_NAME = 'Mikel';
    const LAST_NAME  = 'King';
    const DEFAULT_TZ = 'America/New_York';
    const STATUS     = 'ALIVE';
    const EYE_COLOUR = 'Hazel';
    const ALLERGIES  = 'Chocolate';
}
```



# Inheritance in people

```
<?php
```

```
class Great_Grand_Mother extends Great_Great_Grand_Father {  
    const FIRST_NAME = 'Mary';  
    const LAST_NAME  = 'Luurtsema';  
    const DEFAULT_TZ = 'America/New_York';  
    const STATUS     = 'ALIVE';  
    const ALLERGIES  = 'None';  
}
```

```
<?php
```

```
class Great_Great_Uncle extends Great_Great_Grand_Father {  
    const FIRST_NAME = 'James';  
    const LAST_NAME  = 'Johnson';  
    const DEFAULT_TZ = 'America/New_York';  
    const STATUS     = 'ALIVE';  
    const ALLERGIES  = 'Chocolate';  
}
```



```
<?php
```

```
class Great_Great_Grand_Father {  
    const FIRST_NAME    = 'Howard';  
    const LAST_NAME     = 'Johnson';  
    const DEFAULT_TZ    = 'America/Los_Angeles';  
    const STATUS        = 'Deceased';  
    const HAIR_COLOUR   = 'Dark Brown';  
    const EYE_COLOUR    = 'Dark Brown';  
    const ALLERGIES     = 'Chocolate';  
  
    public static $instance;  
  
    public static $class_name;
```

```
public function print_stats() {
    print( 'Class name: ' . get_class( static::$instance ) . PHP_EOL );
    print( 'Name: ' . static::FIRST_NAME . ' ' . static::LAST_NAME .
PHP_EOL );
    print( 'Hair colour: ' . static::HAIR_COLOUR . PHP_EOL );
    print( 'Eye colour: ' . static::EYE_COLOUR . PHP_EOL );

    print( 'Status: ' . static::STATUS . PHP_EOL );

    $this->print_allergies();
}

public function print_allergies() {
    print( 'Known allergies: ' . static::ALLERGIES . PHP_EOL );
}
```

```
public static function get_instance() {  
    $new_instance = new static();  
    self::set_tz();  
  
    if ( self::$instance === null || ( self::$instance != $new_instance ) ) {  
        self::$instance = $new_instance;  
    }  
  
    return( self::$instance );  
}  
  
public static function set_tz() {  
    date_default_timezone_set( static::DEFAULT_TZ );  
}  
}
```

# Unfortunately People != Programs

- ❖ People are the result of binary inheritance
- ❖ Classes in PHP are unary
- ❖ This example is a single tree

- ❖ All of these classes extend the Great-Great-Grandfather class
- ❖ Great-Great-Grandfather is the root of the tree
- ❖ Great-Great-Grandfather is based upon a simple singleton design pattern
- ❖ These descendant classes work because of late static binding

Late static binding is critical to pulling off some of this magick.

So who wants to look at  
some real code?

Break into live demo or a dance or whatever there's still time for.



Special\_RSS\_Feed => RSS\_Base =>  
Feed\_Base\_Controller => WP\_Base  
=> Base\_Plugin => Singleton\_Base

Special\_Json\_Feed => Json\_Base =>  
Feed\_Base\_Controller => WP\_Base  
=> Base\_Plugin => Singleton\_Base

Twitter:

@MikelKing

GitHub:

<https://github.com/mikelking>

LinkedIn:

<https://www.linkedin.com/in/mikelking>

x1:

mikelking

Once again how to contact me



Destiny!

# Magickal MU Plugins



*Mikel King*

*If you don't use what you make then you failed to  
make something useful*

fin