Framework Variables

Basic Use

Variables defined in Fat-Free are global, i.e. they can be accessed by any component. Framework globals are not identical to PHP globals. An F3 variable named `content` is not identical to PHP's `$content`. F3 is a domain-specific language in its own right and maintains its own separate symbol table for system and application variables. The framework, like every well-designed object-oriented program, does not pollute the PHP global namespace with constants, variables, functions or classes that might conflict with any application. Unlike other frameworks, F3 does not use PHP's `define()` statement. All framework constants are confined to classes.

To assign a value to a Fat-Free variable:

```
$f3->set('var',value)
```

Note: Fat-Free variables accept all PHP data types, including objects and anonymous functions.

To set several variables at once:

```
$f3->mset(
    array(
        'foo'=>'bar',
        'baz'=>123
    )
);
```

To retrieve the value of a framework variable named `var`:

```
$f3->get('var')
```

To remove a Fat-Free variable from memory if you no longer need it (discard it so it doesn't interfere with your other functions/methods), use the method:

```
$f3->clear('var')
```

To find out if a variable has been previously defined:

```
$f3->exists('var')
```

You can also watch a video (https://youtu.be/f3kV7oYG89M) that goes over the main points in this variable user guide.

Globals

F3 maintains its own symbol table for framework and application variables, which are independent of PHP's. Some variables are mapped to PHP globals. Fat-Free's `SESSION` is equivalent to `$_SESSION`, and `REQUEST` maps to `$_REQUEST`. Use of framework variables is recommended, instead of PHP's, to help you with data transfer across different functions, classes and methods. They also have other advantages:

- You can use framework variables directly in your templates.
- You don't have to instruct PHP to reference a variable outside the current scope using a global keyword inside each function or method. All F3 variables are global to your application.
- Setting the Fat-Free equivalent of a PHP global like `SESSION` also changes PHP's underlying `$_SESSION`. Altering the latter also alters the framework counterpart.

Fat-Free does not maintain just a dumb storage for variables and their values. It can also automate session management and other things. Assigning or retrieving a value through F3's `SESSION` variable auto-starts the session. If you use `$_SESSION` (or session-related functions) directly, instead of the framework variable `SESSION`, your application becomes responsible for managing sessions.

As a rule, framework variables do not persist between HTTP requests. Only `SESSION` and `COOKIE` (and their elements) which are mapped to PHP's `$_SESSION` and `$_COOKIE` global variables are exempt from the stateless nature of HTTP.

There are several predefined global variables used internally by Fat-Free, and you can certainly utilize them in your application. Be sure you know what you're doing. Altering some Fat-Free global variables may result in unexpected framework behavior.

The framework has several variables to help you keep your files and directory structures organized. We've seen how we can automate class loading by using the `AUTOLOAD`. There's a `UI` global variable, which contains the path pointing to the location of your HTML views/templates. `DEBUG` is another variable you'll be using quite often during application development and it's used for setting the verbosity of error traces.

Refer to the Quick Reference (quick-reference) for a comprehensive list of built-in framework variables.

Naming Rules

A framework variable may contain any number of letters, digits and underscores. It must start with an alpha character and should have no spaces. Variable names are case-sensitive.

F3 uses all-caps for internal predefined global variables. Nothing stops you from using variable names consisting of all-caps in your own program, but as a general rule, stick to lowercase (or camelCase) when you set up your own variables so you can avoid any possible conflict with current and future framework releases.

You should not use PHP reserved words like `if` , `for` , `class` , `default` , etc. as framework variable names. These may cause unpredictable results.

Working with String and Array Variables

F3 also provides a number of tools to help you with framework variables.

```
$f3->set('a','fire');
$f3->concat('a','cracker');
echo $f3->get('a'); // returns the string 'firecracker'

$f3->copy('a','b');
echo $f3->get('b'); // returns the same string: 'firecracker'
```

F3 also provides some primitive methods for working with array variables:

```
$f3->set('colors',array('red','blue','yellow'));
$f3->push('colors','green'); // works like PHP's array_push()
echo $f3->pop('colors'); // returns 'green'

$f3->unshift('colors','purple'); // similar to array_unshift()
echo $f3->shift('colors'); // returns 'purple'

$f3->set('grays',array('light','dark'));
$result=$f3->merge('colors','grays'); // merges the two arrays
```

Do-It-Yourself Directory Structures

Unlike other frameworks that have rigid folder structures, F3 gives you a lot of flexibility. You can have a folder structure that looks like this (parenthesized words in all-caps represent the F3 framework variables that need tweaking):

```
/   your Web root, where index.php is located
    app/   application files
        dict/   LOCALES, optional
        controllers/
        logs/   LOGS, optional
        models/
        views/   UI
    css/
    js/
    lib/   you can store base.php here
    tmp/   TEMP
        cache/   CACHE
```

Feel free to organize your files and directories any way you want. Just set the appropriate F3 global variables. If you want a really secure site, Fat-Free even allows you to store all your files in a non-Web-accessible directory. The only requirement is that you leave `index.php`, `.htaccess` and your public files, like CSS, JavaScript, images, etc. in a path visible to your browser.

About the F3 Error Handler

Fat-Free generates its own HTML error pages, with stack traces to help you with debugging. Here's an example:

If you feel it's a bit too plain or wish to do other things when the error occurs, you may create your own custom error handler:

```
$f3->set('ONERROR',
    function($f3) {
        // custom error handler code goes here
        // use this if you want to display errors in a
        // format consistent with your site's theme
        echo $f3->get('ERROR.text');
    }
);
```

F3 maintains a global variable containing the details of the latest error that occurred in your application. The `ERROR` variable is an array structured as follows:

```
`ERROR.code` - the HTTP status error code (`404`, `500`, etc.)
`ERROR.status` - a brief description of the HTTP status code. e.g. `'Not Found'`
`ERROR.text` - error context
`ERROR.trace` - stack trace stored in an `array()`
`ERROR.level` - error reporting level (`E_WARNING`, `E_STRICT`, etc.)
```

If your project uses templates, you may wish to handle potential template errors by clearing any existing output buffer and display a fresh page instead:

```
$f3->set('ONERROR',
    function($f3) {
    // recursively clear existing output buffers:
        while (ob_get_level())
            ob_end_clean();
        // your fresh page here:
        echo $f3->get('ERROR.text');
    }
);
```

While developing your application, it's best to set the debug level to maximum so you can trace all errors to their root cause:

```
$f3->set('DEBUG',3);
```

Just insert the command in your application's bootstrap sequence.

Once your application is ready for release, simply remove the statement from your application, or replace it with:

```
$f3->set('DEBUG',0);
```

This will suppress the stack trace output in any system-generated HTML error page (because it's not meant to be seen by your site visitors).

`DEBUG` can have values ranging from `0` (stack trace suppressed) to `3` (most verbose with class and function call logs).

Don't forget! Stack traces may contain paths, file names, database commands, user names and passwords. You might expose your Web site to unnecessary security risks if you fail to set the `DEBUG` global variable to `0` in a production environment.

It is possible to customize the error description at the time you trigger the error, by passing the new description as a second argument, like such :

```
$f3->error(401, "The information necessary to grant access is missing from the reques
t.");
```

Configuration Files

If your application needs to be user-configurable, F3 provides a handy method for reading configuration files to set up your application. This way, you and your users can tweak the application without altering any PHP code.

There are 4 predefined section names:

- `[globals]` for global variables definitions
- `[routes]` for routes definitions
- `[maps]` for route maps definitions
- `[redirects]` for redirecting routes

NB: `[globals]` is assumed if no section has been provided. You can combine all sections in a single configuration file - although having `[routes]`, `[maps]`, and `[redirects]` in a separate file is recommended. This way you can allow end-users to modify some application-specific flags, and at the same time restrict them from meddling with your routing logic.

Here's how to use the different sections:.

[globals]

Instead of creating a PHP script that contains the following sample code:

```
$f3->set('num',123);
$f3->set('str','abc');
$f3->set('hash',array('x'=>1,'y'=>2,'z'=>3));
$f3->set('items',array(7,8,9));
$f3->set('mix',array('this',123.45,FALSE));
```

You can construct a configuration file that does the same thing:

```
[globals]
num=123
; this is a regular string
str=abc
; another way of assigning strings
str="abc"
; this is an array
hash[x]=1
hash[y]=2
hash[z]=3
; dot-notation is recognized too
hash.x=1
hash.y=2
hash.z=3
; this is also an array
items=7,8,9
; array with mixed elements
mix="this",123.45,FALSE
```

Instead of lengthy `$f3->set()` statements in your code, you can instruct the framework to load a configuration file as code substitute. Let's save the above text as setup.cfg. We can then call it with a simple:

```
$f3->config('setup.cfg');
```

String values need not be quoted, unless you want leading or trailing spaces included. If a comma should be treated as part of a string, enclose the string using double-quotes - otherwise, the value will be treated as an array (the comma is used as an array element separator). Strings can span over several lines:

```
[globals]
str="this is a \
very long \
string"
```

To see a video regarding configuration files and some basic code organization, you can check out this video (https://youtu.be/1emj-H4Re-o).

[routes]

F3 also gives you the ability to define HTTP routes in configuration files:

```
[routes]
GET /=home
GET /404=App->page404
GET /page/@num=Page->controller
; Cache the route for 10 minutes
GET /contact=App->contact, 600
; named route
GET @about: /about=Page->about
```

[maps]

Route maps can be defined in configuration files too:

```
[maps]
/blog=Blog\Login
/blog/@controller=Blog\@controller
```

[redirects]

You can also redirect obsolete routes to new pages in configuration files:

```
[redirects]
GET|HEAD /obsoletepage = /newpage
GET|HEAD /dash = @dashboard
GET|HEAD /search = https://www.google.com
```

[configs]

You can also include other config files.

```
[configs]
app/routes.ini = true
app/app_ext.ini = false
```

Don't let the right value confuse you, both files above are included. The `true` / `false` boolean triggers the `$allow` argument on the config (base#config) method, which enables to resolve dynamic token in your config files.

Custom sections

Any other section name than the 3 above is interpreted as a `[globals]` section prefixed by the section name.

So the following:

```
[foo]
a = 1
b = 2

[foo.hash]
x = 1
y = 2
z = 3
```

is equivalent to:

```
$f3->set('foo',array(
  'a' => 1,
  'b' => 2,
  'hash' => array(
    'x' => 1,
    'y' => 2,
    'z' => 3
  )
));
```

Section Hooks

It's also possible to define a custom callback that is applied to every value:

```
[foo.bar:strtoupper]
x = hello
y = world
```

← 2. Routing Engine (routing-engine)

4. Views and Templates → (views-and-templates)