Alternative Hungarian Notation

Alternative Hungarian Notation

http://en.wikibooks.org/wiki/PHP_Programming/Alternative_Hungarian_Notation
This Book Is Generated By Wb2PDF
using
RenderX XEP, XML to PDF XSL-FO Formatter

11-05-2011

Table of Contents

Alternative Hungarian Notation

How To Get Started with PHP Alternative Hungarian Notation

Hungarian Notation is a programming language variable naming convention. Since around 1999 when Charles Simonyi, who originated from Hungary, introduced the naming convention, some have tried to adapt it to various new programming languages. It helps one not only understand what the variable is for, but the intended data type inside it as well.

For PHP, the **PHP Alternative Hungarian Notation** (or **PAHN**) is an attempt at setting forth a naming convention for PHP based on Hungarian Notation, but in a more simplified format, and one that addresses difference in the PHP language from the one that Simonyi was using.

Benefits

- By using a naming convention for variables that is different than functions, class methods, or class variable names, it helps make the code more readable so that you don't confuse a variable for a function call, for instance.
- It helps other programmers coming back to your project understand the intent of that variable.
- By sticking to a simple standard like PAHN that is easy to learn, it is one measure (of many) to make the code from several team members look identical. Otherwise, one team member may use one variable naming convention, while another team member may use another.
- There are not a lot of published standards for PHP-based variable naming conventions. By having one set down here, it is one opportunity to settle the issue, and to have a central document on the web to which many can refer.
- Class variables used for model objects, such as $Member, need to stand out more so than $nMember or $sMember, for instance. Using a naming convention helps improve readability for separating the two.
- Imagine we want to delineate that $sMemberID means an ID that may be alphanumeric, while $nMemberID is going to be an integer. If we do $MemberID, you don't pick up on that so easily. So, again, this naming convention improves readability.
- If we were to use $NamesArray, it's more typing than $asNames. Plus, we have no idea with $NamesArray whether it's an array of Names objects, an array of Names' strings,

or what. So, again, we can improve readability of the code by using this naming convention.

Guidelines

The PAHN variable naming convention begins with a series of prefix characters, followed by a ProperCase variable name. Example:

```
 global $gasNames;
 $gasNames = $Members->getNames();
```

The *$gasNames* would mean global + array + string, or global array of strings.

The prefixes are:

```
_ = a private class variable
a+ = array (often combined with the data type used inside the array)
c+ = character
s+ = string
o+ = object
d+ = date object -- as in what's returned from a date() or gmdate()
v+ = variant -- used very infrequently to mean any kind of possible variable
 type
n+ = numeric (doesn't matter whether it's float, currency, integer, etc.)
x+ = to let other programmers know that this is a variable intended to be used
 by reference rather than value
rs+ = db recordset (set of rows)
rw+ = db row
h+ = handle, as in db handle, file handle, connection handle, curl handle,
socket handle, etc.
g+ = global var (and used sparingly, and often combined with the datatype used
 for the variable)
b+ = boolean
```

Some examples:

```
 $oMember -- a Member object
 $hFile -- a handle to a file, for instance as passed from the fopen() statement

 $cFirst -- first character retrieved from a string
 $rsMembers -- records of Members, as returned from a database table
 $rwMember -- a single Member record from the database
 $bUseNow -- a boolean flag
 $sxMemberName -- a byref string variable for a name of a member.
 $nCounter -- a numeric counter
 $dBegin -- a beginning date
 $sFirstName -- a string to represent someone's first name
```

```
 $_hDB -- a private class variable to store a database connection handle (often
 addressed by $this->_hDB)
```

For class variable names, PAHN does not use this prefix. Thus you might see something like:

```
 $Members = new Members();
```

For constants, PAHN uses just an uppercase word like MEMBER, and this is often used with only inserting variables in PHP Alternative Syntax or CCAPS.

As for what comes after the prefix, it is preferred to stick with ProperCase, as in $sMemberName rather than $sMEMBERNAME, $sMember_Name, $s_Member_Name, or $smemberName. There are several reasons for this. In the case of $sMEMBERNAME, it would imply that the variable is to be treated like a constant (where uppercasing is often seen), when it is not. In $sMember_Name, it makes for more unnecessary typing, as does $s_Member_Name. And $smemberName runs the s+ prefix against the word "member" and makes for a confusing variable name. Now, saying this, there are some rare exceptions where adding an underscore does help with readability, and in those cases it can be used with PAHN. A good example of this rare exception is with an acronym like FIFO. So, $bFIFOIndicator might be more confusing than $bFIFO_Indicator and the latter would be more preferred.

There is also an exception to this prefix for variables used as loop iterators. Many programmers may be familiar with the short variables: $a, $b, $c, $d, $i, $x, $y, $z used in many textbooks. These are great for when you want to have an iterator variable that you address in a loop and use within arrays. For example:

```
$asMemberNames = array();
for ($i = 1; $i <= 10; $i++) {
  $asMemberNames[$i-1] = $Member->getMemberByID($i);
}
```

Therefore, this exception for loop iterators is allowed because it saves time with less typing, and does not reduce readability.