#### https://sites.google.com/site/yacoset/Home/naming-tips

**Naming Tips**

|  |
| --- |
| Choosing the right name is everything. The correct naming of things makes it easier to learn a system, resolves ambiguities, dissolves confusion, promotes insight, freshens your breath and makes you attractive to the opposite sex.  **Methods, Classes and Variables**  Coming up with the right name for a function can be just as important as what that function does.   * **Do not name methods "ProcessData()"** You only get to use this method name once per career, because you should have been fired immediately afterwards. **Be specific** about what it's doing inside; call it ValidateUserCredentials or EliminateDuplicateRequests or ComputeAverageAge, etc. * **Use naming to help you design the program** Pretend there's a rule saying "you can never write a void function", then think about all the steps your program makes to transform input into output, then chose names for those steps so you could make a written sentence with them. These are now your function names and the sentence is your program's structure * **If it's general, it better be generalized** If you name a class FilterCriteria but it's really used for filtering files then it should be called FileFilterCriteria, even if your program only works on files. FilterCriteria would be the name of an abstract class * **Avoid discussing hard work** This is a matter of style, but in the above I used ValidateUserCredentials and EliminateDuplicateRequests, and yet that sounds like a lot of hard work. Consider naming functions that work on collections as if they were properties, like ValidatedUsers and DistinctRequests to make it sound as if the work has already been done. These could then be rewritten as dynamic properties and make the glue code easier to read, but also make you think differently: the program starts to feel more reactive and functional * **Avoid class names that end with "Manager"** They make you think of basic algorithms as if they were complicated black-boxes. "Management" is the statistical allocation of resources to minimize risk and improve performance, not sorting lists or keeping connections open. Consider SortedList or ConnectionPool * **Use singular names for enumerations** An enumeration/Enum is a list of all possible values for a single entity. Call it animalType instead of animalTypes. **Exception:** combinable enums used as flags (enums assigned powers-of-two values and combined with a bitwise or), like "BindingFlags" or "RegexOptions". In this case using a plural name helps the programmer remember that they can be combined * **Avoid superlatives** SuperCollection, FlexiRecord, MegaListView, UltraResolver etc. are wrong. One day you'll create a new class that does one more thing, and then you'll go groping around for another superlative. After a while you won't remember which does what * **Hungarian notation should encode meaning, not type** [Hungarian notation](http://en.wikipedia.org/wiki/Hungarian_notation)--if you chose to use it at all--was meant to tell you about its intended use, not what its datatype is. Don't use intCounter or strFirstname, instead use usFirstname for **u**n**s**afe user input, sFirstnamefor **s**anitized input, and [things like that](http://www.joelonsoftware.com/articles/Wrong.html) * **Don't hide behind your names** Let's say you clean user input by converting it to UTF-8, normalizing entities and escaping the quote marks. Don't do this all in the Escape() method; you need a ToUTF8() method, a NormalizeEntities() method, and then an Escape() method. If you want the convenience of a single method that does it all then make sure it has a vague name that suggests it's doing many separate things, like SanitizeInput() * **Don't forget to have a vowel movement** "f u cn rd ths u cd b a cmptr prgrmmr". You'd also be wearing glasses after three years, and a nervous twitch after twelve (trust me). Don't be afraid to spell things out properly; this isn't the era of 40-character terminals anymore * **Consistency, Consistency, Consistency** Merely being consistent replaces volumes of documentation and months of training--it's literally that powerful when applied to a company, department, or body of code. Your documentation can be more concise, and once a programmer has learned one program or module he can intuit *all* of them with little additional training * **Don't be afraid to rename** Sometimes you'll realize your first choice of name was inappropriate, but if it isn't used in a published API then you should strongly consider renaming it even if your development environment doesn't make it easy   **Frameworks & Libraries**  Your user is a programmer and his biggest problem is discovery. Before documentation, Namespaces are the first tool to solve this problem.   * **Prefer namespaces instead of "nsStupidClassPrefix"** Hopefully your language has namespace support, so use it instead of class name prefixes. If your language doesn't have namespaces then go to the developer community and find out what conventions they use before you invent your own--or else you'll make another PHP * **Extend a generic namespace, don't create MyCompanyName.Widgets** Creating a root namespace named after your corporation will haunt you. Companies and brands get renamed, acquired, sued, and so-on. Bonjour was once Rendezvous, Cocoa was once NeXTstep. Oracle's Vending Machine division was once Sun Microsystems   **Databases**  Database schemas are meant to model data, so the names you choose need to make sense in the context of the domain and *not* for the convenience of the programmer.   * **Use pluralized names for tables** Don't make them singular because you have an ancient ORM and you want the class names to be singular, too. A table contains many rows so its name should be pluralized. Call them "items", "customers", "journalEntries" and so-on, and get a better ORM * **Use aux\_ and meta\_ for tables that contain derived data or housekeeping junk1** One should assume that any table contains "real" data, but if it contains some kind of derived data or system housekeeping trivia then you should make it clear that they're different in purpose * **Name surrogate primary keys after the table** If your table is called "driverLicenses" and needs an ID column for its primary key, then call it "driverLicense\_id" instead of just "id". When you do this for surrogate keys you're implicitly giving it a type, so that "driverLicense.id" is not the same thing as "vehicle.id". It also makes it clear what it is in result sets without needing to use aliases, because if you do use an alias in an app that reads-and-updates then the app would need some way to know the alias isn't the real name of the column * **Use a postfix to show the kind of key** These are used to communicate meta-data that can't be easily encoded in the type, domain or constraints, such as ISBN and Dewey Decimal numbers, social-security numbers, VINs, and so-on. From Joe Celko's [SQL Programming Style](http://www.amazon.com/gp/product/0120887975?ie=UTF8&tag=synesmedia-20&linkCode=as2&camp=1789&creative=390957&creativeASIN=0120887975): \_id for surrogate identities (GUIDs, Identity/Sequence numbers) \_nbr for a string of digits that follows a rigorous standard (driver's license numbers, social-security numbers, etc.) \_code for standardized codes (eg: Zip codes, ISO country codes) \_cat for category names \_class for subclassifications \_type for less-formal class names, such as "motorcycle", "automobile", and "taxi" for driver's license types   1 you can use whatever prefix/suffix you like, so long as you use a prefix that your junior programmers will grok easily  **Things**   * **Name physical things what they are, not what they're doing** They set aside the end-caps of the store aisles to display candy, so you used "c"-for-candy as a postfix for the aisle name in the database. Now they want to put ice-cream and harlequin romance novels there instead. You should have used "e"-for-end-cap. Name real things what they *are* * **Name logical things after what they're doing, not what they are2** Your program uses a table to guide the automatic allocation of new products to shelves by type, but there's no rule that says they can't override and mix product types on the same shelf (they want to stock Star Wars candy next to Star Wars breakfast cereal). You called the table "shelf\_types", but now everyone looks at the table and thinks they'll break the program if they mix the stock. You should have called the table "shelf\_assignment\_guides". Name imaginary things after what they do * **Avoid the "Category" problem** Don't use "Category" as an attribute name, because you'll soon find yourself needing to track different kinds of category and nothing to name the attribute with. Soon you'll have "type" and "kind" and "variant" and "classification" and "subcategory" and nobody will know which is used for what. Look for well-known standards like the Dewey system. Implement a tagging system. And think of specific classifications, like "FuelEfficiencyGrade", "PackagingType", "AgeGroup", "Flamability", "AllergenLevel", etc.   2 but you should use context to simplify your names, too. Eg: your database has two domains, one named guidance, and the other inventory. |