

MimerDesk Coding Guidelines

Teemu Arina <teemu@ionstream.fi>

v0.2, 10 September 2001

This document describes the MimerDesk coding guidelines: How to write code that is correctly formatted and follows all of the required guidelines.

Contents

1	Code Layout	1
2	Coding issues	3
3	Security issues	4
4	Using other programming languages than Perl	4
4.1	Using other programming languages for creating MimerDesk applications	4
4.2	Using Java-applets and other client side browser-based technology	5
4.3	Javascript and DHTML	5
4.4	HTML	5
4.5	CSS	5
5	Recommended reading	5

1 Code Layout

There are some general guidelines for formatting that will make your programs easier to read, understand, and maintain for other MimerDesk developers. If we all follow the same formatting guidelines it is easier to start reading code written for MimerDesk.

- Line up the curly brackets

```
if ($user eq 'sysadmin')
{
    # do stuff
}
```

- No space before the semicolon
- Line up corresponding items vertically

```
$configuration = 'mimerdesk.cfg';
$formatting    = 1;
$debug         = undef;

if ($debug)     {debug();}
elsif ($formatting) {format();}
else           {exit;}
```

- While short identifiers like \$gotit are probably ok, use underscores to separate words in longer ones. It is generally easier to read \$var_names_like_this than \$VarNamesLikeThis
- Global variables are written in uppercase(capital letters): \$VERSION
- Predefine all subroutines in the beginning of the file. This makes it easier to check what routines are defined in the program
- If you have many subroutines classify them in the predefined list. Here is an example:

```
# Messages
sub reply_message;
sub delete_message;
sub edit_message;
sub add_message;

# Forums
sub add_forum;
sub edit_forum;
sub remove_forum;
```

- If you are using a function that uses a hash as the arguments and you have more than two arguments, put every argument on it's own line and line up the corresponding items
- If you are using a function that uses more than four arguments, put every argument on it's own line and line up the corresponding items
- Instead of lining up the curly brackets put everything on one line if the block contains only one command
- Put a comment box before subroutine definition. Write a description of the routine inside the box

```
#####
# Example subroutine #
#_____#
```

- Don't comment lines in a subroutine. Instead use a numbered list under the comment box that describes the different steps in the subroutine. It's easier to read if you don't put comments right above or next to the code

```
# 1. Read files from the directory
# 2. If the current filename starts with a (.)..
#    2.1 Skip file
# 3. Print filenames
```

- If the coder is more likely to add entries later in a predefined hash, add one final semicolon after the last entry
- Define all private variables in the beginning of the subroutine instead of defining them with the built-in function *my()* every time a variable is first time used
- First the main program, then the subroutines in order to preserve natural hierarchy in your program
- Group subroutines which do the same kind of thing
- If your subroutine or program is very big (over one screen) try to find code that you can move into a new subroutine
- If you have big blocks inside curly brackets (over 15 lines of code) there is a better way to write it. Try to use new subroutines or change the structure
- Use reasonable size variable and subroutine names
- Write all subroutine names in lowercase
- Use descriptive variable names. *\$x* and *\$y* are more difficult to read than *\$user* and *\$password*

2 Coding issues

- *use strict* all the time
- *use CGI::Carp "fatalToBrowser"* when developing or testing your application. Remove it when your application is ready
- Add new configuration options in the configuration file and set defaults for the variables you need instead of defining configuration options inside your application. Although you can specify configuration variables inside your application for internal purposes
- Put all hash keys you are pointing at inside to avoid clashing subroutine names in the future

```
# Wrong
$ref->{hashkey}
# Ok
$ref->{'hashkey'}
# Another example where quotes are not necessary
%hash = (
    SECOND => 1,
    MINUTE => 2,
    HOUR   => 3,
);
```

- If your code / module needs documentation Use POD (check out MimerDesk API library for an example)
- Avoid global variables! Use global variables only if there is a wider use for them
- If you can do it in Perl you don't need external programs

- Use modules from CPAN (www.cpan.org) for more complex tasks if available. There is no point in re-inventing the wheel
- Use function `write_log()` instead of built-in function `warn()`
- Use function `write_log()` for all error, warning and other unexpected messages. Try to log everything that might be useful
- If you have many parameters for a subroutine and coders might have problems remembering the right order of the parameters or there might be optional parameters, implement a system that allows hash keys as the parameters in any order (Check out HTML generating functions in MimerDesk API library)
- If possible, use cross-platform compatible code
- Comment your code where it is necessary for understanding
- **Modularize** is an order. Write re-usable code
- Do **not** include HTML, Javascript or other code inside your script. Use HTML generating functions to do the same job or create new templates
- Read the MimerDesk API library documentation thoroughly to see what functions are available for various coding tasks. Use MimerDesk library functions whenever possible

3 Security issues

- Avoid using any shell commands. Use shell only for fall-back actions or for something absolutely necessary
- Always check values passed for database queries with function `prepare_fordb()`
- Always use function `html_escape()` before printing any text submitted by the user. It is too easy for users to submit highly-interactive and ill-conceived Dynamic HTML if the function is not used
- If it is absolutely necessary to use *back-ticks* or function `eval()`, `open()`, `unlink()`, `glob()`, `umask()`, `exec()` and `system()`, check all variables before executing any of these
- Use function `initialize()` in the top part of your program. It will define a more secure path
- Always check ALL USER INPUT. If a value should be a number (e.g age) then it **has** to be exactly that or it is discarded

4 Using other programming languages than Perl

4.1 Using other programming languages for creating MimerDesk applications

Although in theory it is possible to use almost any programming language for MimerDesk modules and programs, we suggest using Perl for code that is included in the final release. All programs written in another language will be treated as third-party programs.

There are several reasons for this. First, it is difficult to maintain multiple versions of the library in many programming languages. Second, in order to fully take advantage of persistent database connections and other

efficiency boosting features of `mod_perl`, all of the programs have to be written in Perl. It will be faster. Using C for structures that need speed (e.g parsers) is however acceptable. Third, MimerDesk will be more difficult to install and maintain if there are several programming languages used.

4.2 Using Java-applets and other client side browser-based technology

Java-applets are welcome if there is a good reason for them ¹. Good reasons are video conferencing, games, interactive programs etc. where Java is simply the best alternative.

Flash applications should offer additional features, not features that are absolutely necessary. Flash is good for tours, music players, multimedia presentations and maybe for games and such. Other browser plug-ins should be used only for material viewing (e.g. PDF).

4.3 Javascript and DHTML

Javascript is welcome but it should not be used too heavily. Only for layout issues and GUI functionality (mouse over images, form verification etc.). If you write Javascript, make sure it works with all of the commonly used browsers (Konqueror, Mozilla, Netscape 4.x or higher, IE 4 or higher and Opera).

4.4 HTML

HTML code should be 4.01 Transitional compliant and it should look mostly the same with all of the commonly used browsers. Make it as standard as possible.

4.5 CSS

CSS is good but the same rule applies here as it applied for HTML: it should look mostly the same with all of the commonly used browsers. CSS should be CSS Version 2 standard compliant. There is a MimerDesk wide style-sheet in the *files* directory. Add your CSS there if possible instead of adding CSS code directly to the template files.

5 Recommended reading

- `perlstyle` (Perl style tutorial) man-page
- `perlpod` (POD documentation) man-page
- `perlport` (Portability tutorial) man-page
- `perlsec` (Security tutorial) man-page
- <http://www.w3.org/Security/Faq> <<http://www.w3.org/Security/Faq>> - Lincoln Stein's World Wide Web Security FAQ
- <http://www.w3.org> <<http://www.w3.org>> - Official standards organization (HTML, CSS etc..)

¹Since java-applets make the GUI a lot slower due to the client side loading, there should be a really good reason for using them!