Система управления контентом SantaFox™

Руководство по стилю программирования

Версия 1.0

SantaFox developer
Saint-Petersburg, 2008

Оглавление

Введение

Для кого это руководство

Данное руководство создано для разработчиков, планирующих создавать собственные модули, работающие под управлением системы управления контентом SantaFox™.

Разработчиком собственного модуля для системы SantaFox™ может стать любой программист обладающий следующими навыками:

- понимание объектно-ориентированного программирования;
- среднее знание PHP;

Мы рекомендуем любому разработчику, в независимости от его квалификации, ознакомиться с данным руководством и придерживаться его.

О чём это руководство

В данном руководстве содержится перечень требований, и рекомендаций по оформлению программного кода модулей, которых должен придерживаться разработчик модулей для SantaFox™.

Использование одного стиля написания программного кода поможет сделать разрабатываемые Вами решения понятными для других разработчик. Это в свою очередь будет способствовать (или, по крайней мере, не будет мешать) вносить правки в уже существующий код и модифицировать его, тем самым расширяя существующее возможности модуля.

Здесь не рассматривается обоснованность, плюсы и минусы того или иного подхода (стиля) используемого в программирование. Для этих целее существует специальная литература, к которой Вы можете обратиться для более детального и пристального изучения проблемы стилистики программного кода.

Задача данного руководства – унифицировать стилистику программного кода всех разработчиков, создающих модули для SantaFox™.

Отступы, длина строки и выравнивание

При написании и форматировании кода обязательно использование отступов. В качестве отступов используется *мягкая табуляция (soft tabs)*. Один символ табуляции включает в себя 4 символа пробела.

Рекомендуем не использовать строки длиннее 150 символов. В случае необходимости переносить строку с учетом отступов, с тем, что бы схожие логические объекты находились на одной вертикально прямой:

Пример переноса:

```
01.      if ($param == 'time' ||
02.          $param == 'data' ||
03.          $param == 'week')
04.      {
05.          return 1;
06.      }
```

Схожим образом переносятся аргументы функций в том случае, если их достаточно много и размещение в одну строчку проблематично.

Использование математических операторов

Блоки операторов присваивания должны быть выровнены по знаку «=». Между любым математическим оператором и аргументами должны быть пробелы. В случае использования в блоке, в каждой однотипной строке нескольких математических операторов, вертикальное выравнивание производится по каждому из них (от данного правила можно отступать в том случае если в каждой строке блока используются несколько разных операторов, и выравнивание приведет к непомерному увеличению строки).

Пример:

```
01.      $nick        =  $_GET['nick_name'];
02.      $full_name   =  $_GET['full_name'];
03.      $data        =  $_POST['curent_data'];
```

Или:

```
01.      $number      = $_GET['num'];
02.      $count       = $count     + 2;
03.      $price       = $count     * $number;
```

Управляющие конструкции

При описании управляющих конструкций используются единый тип размещения фигурных скобок. Открывающая и закрывающая скобка располагается на отдельных и разных строчках (тип BSD):

Примеры кода класса:

```
01.     class my_class()
02.     {
03.       ……
04.     }
```

Пример кода функции:

```
01.     function my_function()
02.     {
03.           ……
04.     }
```

Пример кода «переключателя»:

```
01.     switch ($action)
02.     {
03.           case 'action1':
04.                 ……
05.                 Break;
06.
07.           case 'action2':
08.                 ……
09.                 break;
10.     }
```

Пример кода для оператора условия:

```
01.     if ($result)
02.     {
03.           ……
04.
05.     }
```

Пример кода для оператора цикла:

```
01.     while ($i < 10)
02.     {
03.           ……
04.     }
```

При этом, в простых конструкциях *if, while, for u foreach* (когда в теле конструкции один оператор представленный одной строкой) фигурные скобки не используются.

Пример:

```
01.      if (!$result)
02.               return;
03.
04.      $long_i = '';
05.      if ($i < 256)
06.               $long_i = 'byte';
07.      elseif ($i < 65536)
08.               $long_i = 'word';
09.      else
10.               $long_i = 'very long';
```

Правила именования переменных, функций, таблиц и классов

Имена должны быть максимально простыми и при этом максимально отражать смысл хранимых в них значений или выполняемых действий.

При использовании имен, состоящих из нескольких слов, все слова пишутся со строчной (маленькой) буквы и разделены между собой знаком "_" (нижнее подчеркивание).

Временные переменные, используемые в небольших объемах кода должны состоять из одного слова. В качестве числовых переменных используемых при итерации необходимо использовать однобуквенные переменные $i, $j, $k и т.д.

Запрещено использовать глобальные переменные. При написании кода допустимо обращаться только к одной глобальной переменной $kernel

При использовании констант, к ним применяются выше описанные правила с той лишь разницей, что имя константы пишется полностью прописными (большими) буквами.

Согласованность имен

Переменные, используемые для схожих целей должны иметь схожие имена. Переменные, используемые для связи с базой данных должны иметь имена – соответствующие колонкам этой базы данных.

Имена функций классов должны так же иметь схожие названия (начинаться с одного слова) если они относятся к одному логическому объекту. Если функция класса возвращает какое-то значение из базы данных, массива или запрашивает какое-то свойство – то должно использоваться слово "get"; устанавливает какое-то значение (свойство) – то слово «set».

При придумывании названия своим функциям (названия, которые будут видеть администраторы сайта при создании действий из ваших методов) следует использовать слово «сформировать». То есть ваши методы должны называться «сформировать ленту», «сформировать каталог», «сформировать блок». Не используйте слов «показать», «вывести», «отобразить» и тому подобные, что бы эти слова мог использовать администратор сайта для названия своих действий.

SQL Конструкции

При задании имен таблиц и колонок следует придерживаться общего правила именования переменных.

При конструировании SQL запросов следует придерживаться следующих правил:

- разбивать строки запроса по ключевым словам;
- использовать псевдонимы в сложных запросах;

Запрещается использовать функцию *mysql_query()* для выполнения SQL запросов. Для этого необходимо пользоваться функцией *$kernel->runSQL()*

Комментирования кода

Комментирование объектов

Любая функция и переменная класса, равно как и сам класс, должны быть документированы. Документация к каждому объекту должна содержать исчерпывающую информацию об этом объекте, его назначении и функциях.
Описание размещается непосредственно перед объявлением класса, функции и переменной.

Описание к классу имеет следующую структуру:

```
01.      /**
02.       * Краткое описание класса
03.       *
04.       * Полное описание класса (допустимы HTML теги)
05.       * @name [название класса]
06.       * @package [имя пакета]
07.       * @copyright [разработчик]
08.       * @version [версия]
09.       */
```

@name – имя класса, так как он объявлен в коде

@package – указывается имя пакета, к которому относится данный класс. Программный код может относиться к следующим пакетам:

- Kernel

- Modules
- AdminControl

@copyright – Указывается компания (или физическое лицо), кому принадлежат права на программный код

@version – указывается версия класса. Версия указывается в формате *"[версия класса].[версия реализации]"*. В качестве первой цифры указывается основная версия класса, с определенным набором функций. Вторая цифра показывает версию реализации функций класса. Другими словами, добавлений новых функций должно изменять основную версию класса, а изменения внутри этих функций меняют версию реализации.

Правила определения версий класса являются временными, и подлежат расширению и уточнению с внедрением системы SVN

Описание к переменной внутри класса имеет следующую структуру:

```
01.     /**
02.     * Краткое описание переменной
03.     *
04.     * Полное описание переменной (допустимы HTML теги)
05.     * @access [private или public]
06.     * @var [тип]
07.     */
```

Описание к функции класса имеет следующую структуру:

```
01.     /**
02.     * Краткое описание функции
03.     *
04.     * Полное описание функции
05.     * @param [тип [название] [описание]]
06.     * @access [private или public]
07.     * @return [тип [название] [описание]]
08.     * /
```

В качестве типов могут быть заданы следующие значения:

- string
- integer
- float
- array
- boolean
- resource
- object
- void
- HTML

Если у функции нет параметров, то тег @param не используется, и если функция ничего не возвращает – то тип значения ставится *void.*

@aceess – в данном случае с помощью этого тега происходит определение того, попадает описание данной функции класса (и/или переменной) в автоматически создаваемую документацию или нет. Для большинства функций значение этого параметра должно быть *private* . И только у функции, используемых всеми модулями, а также функций модулей, используемых для создания действий, этот тэг имеет значение *public*.

Как писать полное описание

Полное описание объекта следует давать как можно более подробно, с примерами кода, где используется эта функций. При просмотре данного описание должно быть четкое понимание того, где и для чего применяется данная функция.

При указании примеров исходного кода его необходимо помещать в тег <*code*>:

```
01.      /**
02.      * Функция вывода контента
03.      *
04.      * Перед выводом контента производится его
05.      * проверка на то, что он удовлетворяет всем
06.      * необходимым требованиям.
07.      * Пример вызова функции:
08.      * <code>
09.      *            //Пример вызова
10.      * </code>
11.      * @access private
12.      * @return HTML
13.      * /
```

Если в тексте описание необходимо дать ссылки на другие объекты, с которыми связан описываемый, то необходимо использовать следующую конструкцию:

```
01.      /**
02.      * Функция вывода контента
03.      *
04.      * Полное описание функции вывода контента
05.      * <code>
06.      *            //Пример вызова
07.      * </code>
08.      * Другие ссылки:
09.      *      На объект:      {@link DocExample}.
10.      *      На метод:       {@link DocExample->example()}
11.      *      На переменную: {@link DocExample->$example2}
12.      *      Ещё ссылка:     {@link DocExample->example() метод
   example2()}
13.      * @access private
14.      * @return HTML
15.      * /
```

Комментирование в теле функций

В теле функций так же должны содержаться комментарии. Комментировать необходимо блоки кода, выполняющие то или иное действие. Комментарии внутри кода должны иметь следующий вид:

```
01.      //Инсталлируем значение статистических данных роботов
02.      //из файла
03.      $_list_robot = file("include/install/stat_robot.sql");
04.      for ( $i=0; $i<count($_list_robot); $i++ )
05.      {
06.        <Блок кода>
07.      }
```

Допустимо и блочное комментирование:

```
01.      /*
02.        Инсталлируем значение статистических данных роботов
03.        из файла
04.      */
05.      $_list_robot = file("include/install/stat_robot.sql");
06.      for ( $i=0; $i<count($_list_robot); $i++ )
07.      {
08.        <Блок кода>
09.      }
```

Заключение

Теперь вы можете смело приступать к процессу написания нового модуля для системы SantaFox™, так как ознакомились с требованиями оформления программного кода.

У нас есть уверенность, что соблюдение данного руководства не вызовет у Вас проблем, так как большинство его (руководства) требований являются стандартом де-факто в среде программистов, и вытекают из соображений здравого смысла.