Синтаксические плагины

Синтаксические плагины — это плагины расширения синтаксиса «ДокуВики». Чтобы понять, что необходимо для регистрации нового синтаксиса в «ДокуВики», вы должны прочитать, как работает парсер.

Краткий обзор

Для синтаксического плагина *example* требуется объявить имя класса как `syntax_plugin_<example>`, который расширяет класс `DokuWiki_Syntax_Plugin`, определённый в файле `lib/plugins/syntax.php`. Класс необходимо сохранить в файле с названием `lib/plugins/example/syntax.php`. Для более подробной информации можно обратиться к статье о структуре файлов плагина.

Класс должен содержать как минимум следующие функции:

- ~~`getInfo()` — возвращает хэш с информацией о плагине: автор, электропочта, дата, название, описание, ссылки.~~
  Теперь вместо неё прилагается отдельный файл plugin.info.txt
- `getType()` — должен возвращать тип синтаксиса (см. ниже).
- `getSort()` — возвращает число, по которому определяется, в каком порядке должны добавляться состояния, см. также parser, order of adding modes и getSort list.
- `connectTo($mode)` — эта функция наследуется от класса Doku_Parser_Mode, определённого в `inc/parser/parser.php`. Это место, где регистрируется регулярные выражения, необходимые для опознания вашего синтаксиса.
- `handle($match, $state, $pos, Doku_Handler $handler)` — функция для подготовки совпавшего синтаксиса для использования рендером.
- `render($mode, Doku_Renderer $renderer, $data)` — функция для отображения контента.

Следующие дополнительные методы могут быть переопределены при необходимости:

- `getPType()` — определяет, как данный синтаксис размещается относительно параграфов[1]. Возвращает:
  - `normal` — (значение по умолчанию, используется, если метод не переопределяется) Плагин может использоваться внутри параграфов,
  - `block` — Открытые параграфы должны быть закрыты до вывода плагина или
  - `stack` — Специальный случай. Плагин обёртывает другие параграфы
- `getAllowedTypes()` (значение по умолчанию: `array()`) Должен вернуть массив типов состояний, которые могут быть включены в собственную разметку плагина.
- `accepts($mode)` — Эта функция говорит парсеру, допускает ли плагин синтаксическое состояние $mode в своей разметке. Поведение по умолчанию заключается в проверке наличия $mode в массиве состояний, хранящемся в унаследованном свойстве `allowedModes`.

При необходимости могут быть определены дополнительные функции. Рекомендуется

добавлять впереди символ подчерка к названиям самостоятельно определённых функций для того, чтобы избежать возможных пересечений имён при дальнейшем развитии спецификации плагинов.

Унаследованные свойства

- `allowedModes` — начальное значение: пустой массив, унаследованный от класса `Doku_Parser_Mode`, определённого в `inc/parser/parser.php`. Содержит список других синтаксических состояний, появление которых допустимо внутри собственного синтаксического состояния плагина (т. е. состояния, которые относятся к любым другим элементам разметки «ДокуВики», которые могут быть включены внутрь собственной разметки плагина). Обычно он автоматически заполняется функцией `accepts()`, используя результаты `getAllowedTypes()`.

Синтаксические типы

«ДокуВики» использует различные синтаксические типы для того, чтобы определить, какие синтаксические конструкции могут быть вставлены внутрь другой. Например, вы можете вставить текстовое форматирование внутрь таблицы.

Для того, чтобы интегрировать свой плагин в эту систему, нужно указать, какой тип он имеет и какие типы могут вставляться в него. В настоящий момент доступны следующие типы:

Тип	**Используется в…**	**Описание**
container	listblock, table, quote, hr	Контейнеры — это сложные состояния, которые могут содержать много других состояний, но они не должны использоваться в таблицах и списках (состояние `hr` нарушает общий принцип), поэтому они отнесены к этому типу.
baseonly	header	Некоторые состояния допустимы только внутри базового состояния.
formatting	strong, emphasis, underline, monospace, subscript, superscript, deleted, footnote	Состояния для изменения стиля текста (сноски (footnote) также можно рассматривать как стиль).
substition[2)]	acronym, smiley, wordblock, entity, camelcaselink, internallink, media, externallink, linebreak, emaillink, windowssharelink, filelink, notoc, nocache, multiplyentity, quotes, rss	Состояния, в которых токен просто меняется на что-то; не могут содержать в себе другие состояния.
protected	preformatted, code, file, php, html	Состояния, имеющие начальный и конечный токены, но внутри которых не допустимы никакие другие состояния.
disabled	unformatted	Внутри этого состояния вики-разметка не отрабатывается, переносы строки и пробелы не сохраняются.
paragraphs	eol	Используется для отметки границ параграфов.

Для описания того, что каждый из типов значит и какие другие классы форматирования зарегистрированы в них, читайте комментарии в файле `inc/parser/parser.php`.

Руководство: разбор синтаксического плагина

Цель данного руководства — разобрать концепции, касающиеся синтаксических плагинов «ДокуВики» и пройти шаги, связанные с написанием своего собственного плагина.

Для тех, кто с особым нетерпением жаждет начать: возьмите копию плагина syntax plugin skeleton. Это своего рода костяк — плагин, который выводит «Hello World!», когда встречает токен «<TEST>» в статье вики. Начните наращивать на него «мясо».

Краткая выжимка

состояния — modes

- Каждый отдельный элемент синтаксиса «ДокуВики», включая и ваш собственный плагин, имеет своё собственное состояние.
- Похожие состояния группируются в типы состояний.
- «allowedTypes» каждого состояния определяет, какие другие синтаксические конструкции «ДокуВики» опознаются внутри собственной синтаксической конструкции этого состояния. Все состояния из «allowedTypes» допустимы.
- «Тип» состояния даёт понять другим состояниям, могут ли они допустить использование данного состояния внутри своего синтаксиса.

handle

- Метод `handle()` вызывается, когда парсер решит, что столкнулся в содержимом статьи вики с куском, относящимся к синтаксису вашего состояния.
- Параметр `$state` говорит, какой тип шаблона из приписанных к вашему состоянию сработал. Если это просто обычный текст, то параметр `state` будет установлен в `DOKU_LEXER_UNMATCHED`.
- Производите как можно больше обработки и принятия решений именно тут, оставляя как можно меньше на обработку методу `render()`, потому что выдача метода handle кэшируется. This also means that you shouldn't do any stuff here that mustn't be cached.

render

- Метод `render()` выполняет инструкции отображения, которые применимы к синтаксическому состоянию плагина (и которые были созданы методом `handle()` плагина.
- Добавляйте содержание к выходному документу с помощью `$renderer->doc .= 'content';`
- Убедитесь, что любая выдача вашего плагина **безопасна** (run raw wiki data through an entity conversion function).
- Совершайте как можно меньше операций по обработке и принятию решений здесь, это всё следует делать в методе `handle()`.

Нет никакой гарантии, что метод `render()` будет вызван в то же время, что и метод `handle()`. Инструкции, произведённые хэндлером, кэшируются и могут использоваться рендером в более позднее время. Единственный надёжный способ передать данные от `handle()` к `render()` — это использовать возвращаемый методом `handle()` массив, который передаётся методу `render()` в качестве параметра `$data`.

Ключевые концепции

Состояния

Состояния (более точно синтаксические состояния) — это основа, на которой базируется парсер «ДокуВики». Каждый отдельный элемент разметки «ДокуВики» имеет своё синтаксическое состояние. Например, существует состояние `strong` для работы со **strong**, состояние `superscript` для работы с ^{superscript}, состояние `table` для работы таблицами и многие другие.

Когда парсер сталкивается с разметкой, он попадает в соответствующее этой разметке синтаксическое состояние. Свойства и методы конкретного синтаксического состояния управляют тем, как ведёт себя парсер, пока он в этом состоянии, включая:

- какие другие состояния могут произойти;
- какие инструкции подготовить для рендера.

Ваш плагин добавит своё синтаксическое состояние к парсеру — это автоматически производится «ДокуВики», когда впервые загружает плагин, назначаемое имя — `plugin_`+ имя директории плагина (которое является также именем класса плагина без префикса «`syntax_`»). Затем, когда парсер сталкивается с разметкой, используемой вашим плагином, он (парсер) войдёт в это синтаксическое состояние. Пока он находится в этом состоянии, ваш плагин управляет тем, что может делать парсер.

Типы состояний

Для упрощения синтаксические состояния, которые ведут себя одинаковым образом, были сгруппированы в несколько типов состояний — полный список может быть найден в разделе «Синтаксические плагины».

Каждый тип состояний соотносится с ключом в массиве $PARSER_MODES. Элемент этого массива, соответствующий каждому типу состояний, сам является массивом, который содержит все синтаксические состояния, относящиеся к этому типу. Например, в «чистой» «ДокуВики» без установленных плагинов элемент массива `$PARSER_MODES['formatting']` содержит: strong, emphasis, underline, superscript, subscript, monospace, deleted и footnote.

Когда плагин загружается в парсер, то через `getType()` он запрашивается о типе состояния, к которому относится. Затем синтаксические состояния, относящиеся к плагину, добавляются в соответствующий массив $PARSER_MODES.

 Указанный вашим плагином тип состояний определяет, где в статье «ДокуВики» парсер будет опознавать разметку вашего плагин. Другие синтаксические состояния «ДокуВики» (также, как и плагины) не будут знать о вашем плагине, но они знают о различных типах состояний.

Если они допускают конкретный тип состояний, он допускают все состояния этого типа, включая любые плагины, которые заявили этот тип состояний.

Выберите тип состояний для своего плагина, сравнивая поведение своего плагина с поведением стандартных состояний «ДокуВики». Выберите тип, к которому относятся наиболее похожие состояния.

Допустимые состояния

Есть другие состояния, которые могут возникнуть внутри разметки вашего собственного состояния.

Каждое синтаксическое состояние имеет собственный массив допустимых состояний, который говорит парсеру, какие именно другие синтаксические состояния будут опознаваться во время обработки состояния. То есть, если вы хотите, чтобы ваш плагин мог оказаться внутри разметки «**strong**», тогда состояние `strong` должен включить состояние вашего плагина в свой массив «allowedModes». И если вы хотите позволить разметке `strong` включаться внутрь разметки вашего плагина, то ваш плагин должен содержать `'strong'` в своём массиве «allowModes».

 Ваш плагин собирает в массив «allowedModes» другие синтаксические состояния посредством типа состояний, объявляемого методом `getType()`.

 Ваш плагин сообщает парсеру, какие другие синтаксические состояния он допускает, декларируя (объявляя) их через метод `getAllowedTypes()`.

PType

PType определяет, как парсеру работать с HTML-элементами <p>, когда он имеет дело с вашим синтаксическим состоянием.

Обычно в тот момент, когда парсер сталкивается с некоторой разметкой, имеется открытый HTML-тэг параграфа. Парсеру необходимо знать, должен ли он закрыть этот тэг перед входом в ваше синтаксическое состояние и затем открыть другой параграф на выходе (`PType='block'` или `PType='stack'`) или парсер должен оставить параграф в покое (`PType='normal'`).

PType также определяет будет ли параграф создаваться внутри вашего синтаксического состояния. И если будет - то как.

Если `PType='normal'`, то параграф не будет создаваться вовсе.

Если PType='block', то парсер закроет параграф перед входом в ваше синтаксическое состояние, и откроет новый параграф после выходы из вашего синтаксического состояния. Внутри вашего синтаксического состояния создание параграфов должно быть запрещено (см. "Допустимые состояния"), иначе открытый параграф закроется как обычно (как только встретятся два подряд или более newlines), и откроется новый. В результате, например, открывающий TAG <DIV> может оказаться внутри одного параграфа, а закрывающий </DIV> внутри другого. Или ещё что-то.

Если PType='stack', то парсер:

1. закроет параграф перед входом в ваше синтаксическое состояние;
2. откроет параграф после входа в ваше синтаксическое состояние;
3. если внутри вашего синтаксического состояния разрешёно создавать параграфы (см. "Допустимые состояния"), то параграфы внутри вашего синтаксического состояния будут создаваться как обычно (как только встретятся два подряд или более newlines). Если не разрешено, то параграф так и останется открытым до выхода из вашего синтаксического состояния;
4. перед выходом из вашего синтаксического состояния парсер закроет параграф (в этом месте он должен быть открыт, не зависимо от того, разрешено ли создавать параграфы внутри или нет);
5. после выходы из вашего синтаксического состояния парсер откроет новый параграф.

Для тех, кто знает CSS, возвращение PType='block' означает, что html, произведённый вашим плагином, будет похож на display:block, а возвращение PType='normal' означает html, похожий на display:inline.

Пример

Предполагается, что вы хорошо знакомы со стандартным шаблоном syntax plugin ENTRY ⇒ UNMATCHED ⇒ EXIT. В зависимости от значения PType и будут расставляться автоматически рендером в разных точках снаружи и внутри текста плагина. Поэтому вашему плагину не нужно заботиться об этих тэгах.

wikisyntax	PType=normal	PType=block	PType=stack
foo <plugin>text</plugin> bar	foo ENTRY("<plugin>") UNMATCHED("text") EXIT("</plugin>") bar	foo ENTRY("<plugin>") UNMATCHED("text") EXIT("</plugin>") bar	foo ENTRY("<plugin>") UNMATCHED("text") EXIT("</plugin>") bar

Порядковый номер

Этот номер используется лексером[3)] для управления порядком, в котором он проверяет шаблоны синтаксических состояний на «сырых» (исходных) данных вики. Это важно только в том случае, если один и тот же участок данных попадает в шаблоны, относящиеся к двум или более состояниям. После проверки будет выбран шаблон, относящийся к состоянию с наименьшим порядковым номером.

Вы можете использовать это свойство для написания плагина, который заменяет или расширяет «родной» хендлер «ДокуВики» для той же синтаксической конструкции. Примером является плагин «Code».

Подробности о существующих порядковых номерах доступны для обоих parser (sort list). 

Шаблоны

Парсер использует PHP-функции «preg»[4]. Детальное объяснения регулярных выражений и их синтаксиса выходит за пределы этого руководства. Существует много хороших источников в интернете.

Полный синтаксис «preg» не доступен для использования в конструировании шаблонов синтаксических плагинов. Ниже приведён список известных различий:

- Шаблоны не окружаются разделителями.
- Для использования вертикальной черты «|» при множественных альтернативах, сделайте их non-captured-группами, т. е. «(?:cat|dog)».
- Будьте очень осторожны с «заглядыванием назад». Парсер только пытается сравнить шаблон со следующим куском «ещё не проверенных» данных. Если вам нужно заглянуть в символы, которые уже участвовали в предыдущем сравнении с шаблоном, то этих символов там на самом деле не будет.
- Флаги опций могут быть включены только как встроенные опции, т. е. `(?i)`, `(?-i)`.

Парсер предоставляет плагину четыре функции для регистрации необходимых шаблонов. Каждая функция относится к шаблонам с разными смыслами.

- **специальные шаблоны** — `addSpecialPattern()` — это шаблоны, которые используются, когда один шаблон — это всё, что нужно. В терминах парсера эти шаблоны представляют и вход в синтаксическое состояние плагина, и выход из этого синтаксического состояния, всё в одно сравнение. Обычно они используются в плагинах `substition`.
- **входные шаблоны** — `addEntryPattern()` — шаблон, указывающий на начало данных, которые должны быть обработаны плагином. Обычно эти шаблоны должны включать в себя заглядывание вперёд для проверки существования выходного шаблона. Любой плагин, который регистрирует входной шаблон, также должен зарегистрировать выходной шаблон.
- **выходные шаблоны** — `addExitPattern()` — шаблон, указывающий на конец данных, которые должны быть обработаны плагином. Этот совпадение с этим шаблоном может произойти, только если было найдено совпадение с входным шаблоном[5].
- **внутренние шаблоны** — `addPattern()` — представляют специальный синтаксис, применимый к плагину, который может встретиться между входным и выходным шаблонами. Обычно это нужно только для достаточно сложных структур, например, таблиц и списков.

Один плагин может добавить несколько шаблонов в парсер, включая более чем один шаблон

одного типа.

Советы

- Используйте «нежадные» идентификаторы, т. е. +? или *? вместо + или *.
- Будьте осторожны с использованием нескольких выходных шаблонов. Скорее всего сработает выход из состояния по первому встреченному выходному шаблону, даже если это будет не тот шаблон, который «увидел при заглядывании вперёд» входной шаблон. Необходимость нескольких выходных шаблонов может означать, что на самом деле вам нужны несколько плагинов.
- В ранних версиях «ДокуВики» лексер имел баг, не дававший использовать угловые скобки «<» или «>» в «заглядывающих вперёд» шаблонах. Этот баг был исправлен и угловые скобки теперь допустимы. Некоторые плагины всё ещё содержат шестнадцатеричные обозначения для угловых скобкок («\x3C», «\x3E») — такой приём позволял обойти баг.

Метод handle()

Это часть вашего плагина, которая должна совершать всю работу. До того как «ДокуВики» выведет статью вики, он создаёт список инструкций для рендера. Метод `handle()` плагина создаёт инструкции отображения для собственного синтаксического состояния. В некий более поздний момент они будут интерпретированы методом `render()` плагина. Список инструкций кэшируется и может быть использован много раз, разумно максимально увеличить объём работы, совершаемой один раз этой функцией и максимально уменьшить объём работы, совершаемый много раз функцией `render()`.

Параметр `$match` — текст, который совпадает с шаблоном, или, в случае `DOKU_LEXER_UNMATCHED`, непрерывный кусок обычного текста, который не совпал с каким-либо шаблоном.

Параметр `$state` — тип шаблона, из-за которого запустился вызов handle().

- `DOKU_LEXER_ENTER` — шаблон установлен функцией addEntryPattern();
- `DOKU_LEXER_MATCHED` — шаблон установлен функцией addPattern();
- `DOKU_LEXER_EXIT` — шаблон установлен функцией addExitPattern();
- `DOKU_LEXER_SPECIAL` — шаблон установлен функцией addSpecialPattern();
- `DOKU_LEXER_UNMATCHED` — обычный текст, встреченный внутри синтаксического состояния плагина, который не совпал ни с одним шаблоном.

Параметр `$pos` — позиция первого символа найденного текста.

Параметр `&$handler` — ссылка на объект Doku_Handler.

Метод render()

Часть плагина, которая производит вывод окончательной веб-страницы или какой-либо другой поддерживаемый формат. Именно здесь плагин добавляет собственный вывод к уже созданным другими частями рендерера путём склейки со свойством `doc` рендера. Т. е.:

```
$renderer->doc .= "некий вывод плагина...";
```

 В любых сырых данных вики, которые передаются `render()`, все спецсимволы должны быть преобразованы в элементы HTML. Вы можете использовать PHP-функции `htmlspecialchars()`, `htmlentities()` или собственный метод `xmlEntities()` рендера. Т. е.:

```
$renderer->doc .= $renderer->_xmlEntities($text);
```

Параметр `$mode` — имя формата состояния финального вывода произведённого рендером. В настоящее время «ДокуВики» поддерживает только один формат вывода — XHTML [6].

Новые состояния могут быть представлены в плагинах рендера. Плагины должны производить вывод только для тех форматов, которые они поддерживают, это значит, что эта функция должна быть структурирована...

```
if ($mode == 'xhtml') {  // supported mode
  // code to generate XHTML output from instruction $data
}
```

Параметр `$data` — массив, содержащий инструкции, предварительно подготовленные собственным методом `handle()` плагина. Эта функция должна интерпретировать инструкции и выдавать соответствующий вывод.

Безопасность

Сырые данные вики, которые достигли вашего плагина, больше никогда не должны обрабатываться. Никакой дальнейшей обработки не производится над выводом после того, как он покидает плагин. Как минимум, плагин должен убедиться, что в выводе все спецсимволы HTML заменены на HTML-последовательности. Также к извлечённым и используемым внутри данным вики нужно относится с вниманием. См. также статью «Безопасность».

Локализация

Смотрите статьи «Локализация» и «Структура файлов плагина».

Конфигурация

Смотрите статью «Конфигурация».

Использование CSS и JavaScript

Смотрите статью «Структура файлов плагина».

Добавление кнопок к панели инструментов

Для того, чтобы облегчить жизнь пользователям вики, которые установили ваш плагин, следует добавить кнопку в панель инструментов редактора.

См. статьи:

- Плагины действий;
- Панель инструментов.

Написание своего собственного плагина

Ну хорошо, вы решили расширить синтаксис «ДокуВики» своим собственным плагином. Вам придётся разработать, каким будет ваша синтаксическая конструкция и как она будет отображаться в браузере пользователя. Теперь вам необходимо написать сам плагин.

1. Примите решение, как назвать плагин. Возможно вы захотите проверить список доступных плагинов, чтобы убедиться, что такое имя уже не используется.

2. В своей собственной установке «ДокуВики» создайте собственную поддиректорию в директории `lib/plugins/`. Эта директория должна называться также, как ваш плагин.
3. Создайте в новой директории файл `syntax.php`. В качестве отправной точки, можете использовать скелет плагина. Скопируйте его в свою директорию.
4. Отредактируйте этот файл под свои нужды:
    - измените название класса, чтобы он был вида `syntax_plugin_`<название вашего плагина>[7];
    - измените метод `getInfo()`, чтобы он выдавал информацию о вашем плагине;
    - измените метод `getType()`, чтобы он выдавал тип состояний, к которому относится ваш плагин;
    - добавьте метод `getAllowedTypes()` для сообщения всех типов состояний, которые ваш плагин может включать внутрь своей собственной синтаксической конструкции. Если ваш плагин не желает позволять какому-либо состоянию включаться в себя, он может быть выкинут;
    - измените метод `getPType()`, чтобы он выдавал PType, который относится к вашему плагин. Если это `'normal'`, вы можете просто убрать этот метод;
    - измените метод `getSort()`, чтобы он выдавал уникальный номер, проверьте его в списке плагинов;
    - измените метод `connectTo()`, чтобы зарегистрировать шаблон для опознавания вашего синтакиса;
    - добавьте метод `postConnect()`, если ваш синтаксис имеет второй шаблон, для того, чтобы указать, когда парсер должен покинуть ваше синтаксическое состояние.
5. Ну вот, простая часть работы сделана, теперь у вас есть плагин, которой скажет «Hello World!», когда встретит шаблон вашего синтаксиса. Самое время проверить его и убедиться, что шаблон работает как надо — посетите свою вики и создайте статью с синтаксической страницей своего плагина, сохраните её и убедитесь, что «Hello World!» действительно показывается.
6. Напишите свои собственные методы `handle()` и `render()`:
    - если у вас есть входной и выходной шаблоны, не забудьте обработать не совпавшие с шаблоном данные;
    - относитесь к сырым данным вики с подозрением (вниманием) и убедитесь, что все спецсимволы прошли конвертор последовательностей.
7. Протестируйте и добавьте плагин на страницу плагинов «ДокуВики».

Пример 1-й плагина — Now

Когда синтаксическая конструкция этого плагина `[NOW]` встречается в статье вики, текущая дата и время отображается в формате RFC2822.

- Типом является `'substition'`. Мы подставляем временую метку вместо токена `[NOW]`, аналогично смайлам и акронимами. Они также относятся к типу состояний `'substition'`.
- Заполнять «allowedTypes» не требуется, т. к. никакие другие синтаксические конструкции не могут появиться в конструкции `[NOW]`. Т. о. нам не нужен метод `getAllowedTypes()`.
- PType — `normal` — это значение по умолчанию, поэтому нам не нужно определять метод `getPType()`.
- Нет необходимости во входном и выходном шаблонами, только специальный шаблон,

чтобы найти `[NOW]`. Единственная вещь, с которой нужно быть осторожным, это то, что символы «[» и «]» имеют специальное значение в регулярных выражениях, поэтому нам нужно «выключить» (escape) их, создав шаблон `'\[NOW\]'`.
- В нашем случае метод `handler()` не должен ничего делать. Нам не нужно заботиться о специальных состояниях или дополнительных параметрах в нашем синтаксисе. Мы просто вернём пустой массив в качестве инструкций для рендера.
- Всё, что нужно методу `render()` — добавить штамп времени к текущей статье вики — `$renderer->doc .= date('r');`

И вот наш плагин завершён!

syntax.php

```
<?php
/**
 * Plugin Now: Inserts a timestamp.
 *
 * @license    GPL 2 (http://www.gnu.org/licenses/gpl.html)
 * @author     Christopher Smith <chris@jalakai.co.uk>
 */

// must be run within DokuWiki
if(!defined('DOKU_INC')) die();

if(!defined('DOKU_PLUGIN'))
define('DOKU_PLUGIN',DOKU_INC.'lib/plugins/');
require_once(DOKU_PLUGIN.'syntax.php');

/**
 * All DokuWiki plugins to extend the parser/rendering mechanism
 * need to inherit from this class
 */
class syntax_plugin_now extends DokuWiki_Syntax_Plugin {

    function getInfo(){
      return array(
        'author' => 'me',
        'email'  => 'me@someplace.com',
        'date'   => '2005-07-28',
        'name'   => 'Now Plugin',
        'desc'   => 'Include the current date and time',
        'url'    => 'http://www.dokuwiki.org/plugin:tutorial',
      );
    }

    function getType() { return 'substition'; }
    function getSort() { return 32; }
    function connectTo($mode) {
        $this->Lexer->addSpecialPattern('\[NOW\]',$mode,'plugin_now');
    }
    function handle($match, $state, $pos, &$handler) {
```

```
        return array($match, $state, $pos);
    }
    function render($mode, &$renderer, $data) {

      if($mode == 'xhtml'){
          $renderer->doc .= date('r');
          return true;
      }
      return false;
    }
}
?>
```

Замечание: из-за способа, которым «ДокуВики» кэширует страницы, этот плагин будет отображать дату/время для момента, когда был создан кэш страницы. Вам нужно добавить на страницу макрос ~~NOCACHE~~,чтобы отображалось правильное время каждый раз, когда запрашивается страница.

Пример 2-й плагина — Color

Когда встретится синтаксическая конструкция плагина `<color somecolour/somebackgroundcolour>` встречается в статье вики, цвет текста сменяется на «somecolour», а цвет фона — на «somebackgroundcolour»; и оба остаются такими, пока не встретится `</color>`.

- То, что мы делаем, подобно тому, что делает состояние `strong`, его тип — 'formatting', поэтому мы тоже должны использовать этот тип.
- «allowedTypes» должны быть inline состояния - `substition`, `formatting` и `disabled`.
- PType — `normal` — это значение по умолчанию, поэтому нам снова не нужен метод `getPType()`.
- Нам нужны входные и выходные шаблоны. Входной шаблон должен проверять, что есть и выходной шаблон, т. е. `'<color.*>(?=.*?</color>)'`. Выходной шаблон аналогично `</color>`.
- Метод `handle()` должен иметь дело с тремя состояниями: совпадающими с входным и выходным шаблонами и «несовпадающим» для промежуточного текста.
    - Состояние `DOKU_LEXER_ENTER` требует некоторой обработки для извлечения значений цветов текста и фона, они войдут в инструкцию для нашего рендера.
    - Состояние `DOKU_LEXER_UNMATCHED` не требует какой-либо обработки, но нам придётся передать «несовпадающий» текст (в параметре `$match`) методу `render()`, поэтому он войдёт в инструкцию для нашего рендера.
    - Состояние `DOKU_LEXER_EXIT` не требует какой-либо обработки и не имеет никаких особых данных, мы просто должны сделать выходную инструкцию для `render()`.
- Методу `render()` необходимо иметь дело с теми же тремя сотояниями, что и `handle()`.
    - `DOKU_LEXER_ENTER` — отрыть тэг `span` с указанием стиля, использующего значения цветов текста и фона.
    - `DOKU_LEXER_UNMATCHED` — добавить «несовпавший» текст к выходному документу.
    - `DOKU_LEXER_EXIT` — закрыть тэг `span`.

Опять же, всё достаточно очевидно. И вот, что мы имеем:

syntax.php

```
<?php
/**
 * Plugin Color: Sets new colors for text and background.
 *
 * @license    GPL 2 (http://www.gnu.org/licenses/gpl.html)
 * @author     Christopher Smith <chris@jalakai.co.uk>
 */

// must be run within Dokuwiki
if(!defined('DOKU_INC')) die();

if(!defined('DOKU_PLUGIN'))
define('DOKU_PLUGIN',DOKU_INC.'lib/plugins/');
require_once(DOKU_PLUGIN.'syntax.php');

/**
 * All DokuWiki plugins to extend the parser/rendering mechanism
 * need to inherit from this class
 */
class syntax_plugin_color extends DokuWiki_Syntax_Plugin {

    /**
     * return some info
     */
    function getInfo(){
        return array(
            'author' => 'Christopher Smith',
            'email'  => 'chris@jalakai.co.uk',
            'date'   => '2008-02-06',
            'name'   => 'Color Plugin',
            'desc'   => 'Changes text colour and background',
            'url'    => 'http://www.dokuwiki.org/plugin:tutorial',
        );
    }

    function getType(){ return 'formatting'; }
    function getAllowedTypes() { return array('formatting',
'substition', 'disabled'); }
    function getSort(){ return 158; }
    function connectTo($mode) {
$this->Lexer->addEntryPattern('<color.*?>(?=.*?</color>)',$mode,'plugin
_color'); }
    function postConnect() {
$this->Lexer->addExitPattern('</color>','plugin_color'); }


    /**
```

```
     * Handle the match
     */
    function handle($match, $state, $pos, &$handler){
        switch ($state) {
          case DOKU_LEXER_ENTER :
                list($color, $background) = preg_split("/\//u",
substr($match, 6, -1), 2);
                if ($color = $this->_isValid($color)) $color =
"color:$color;";
                if ($background = $this->_isValid($background))
$background = "background-color:$background;";
                return array($state, array($color, $background));

          case DOKU_LEXER_UNMATCHED :  return array($state, $match);
          case DOKU_LEXER_EXIT :       return array($state, '');
        }
        return array();
    }

    /**
     * Create output
     */
    function render($mode, &$renderer, $data) {
        if($mode == 'xhtml'){
            list($state,$match) = $data;
            switch ($state) {
              case DOKU_LEXER_ENTER :
                list($color, $background) = $match;
                $renderer->doc .= "<span style='$color $background'>";
                break;

              case DOKU_LEXER_UNMATCHED :  $renderer->doc .=
$renderer->_xmlEntities($match); break;
              case DOKU_LEXER_EXIT :       $renderer->doc .= "</span>";
break;
            }
            return true;
        }
        return false;
    }

    // validate color value $c
    // this is cut price validation - only to ensure the basic format
is correct and there is nothing harmful
    // three basic formats  "colorname", "#fff[fff]",
"rgb(255[%],255[%],255[%])"
    function _isValid($c) {
        $c = trim($c);

        $pattern = "/^\s*(
            ([a-zA-z]+)|                                #colorname -
```

```
not verified
            (\#([0-9a-fA-F]{3}|[0-9a-fA-F]{6}))|        #colorvalue
            (rgb\(([0-9]{1,3}%?,){2}[0-9]{1,3}%?\))     #rgb triplet
            )\s*$/x";

        if (preg_match($pattern, $c)) return trim($c);

        return "";
    }
}
?>
```

Замечание: никаких проверок на корректность названий цветов или значения RGB не производилось.

Дополнения и Файлы

Ссылка на оригинал статьи

1)
См. Doku_Handler_Block
2)
Да, это неправильное написание, но мы не хотели бы его менять, чтобы избежать нарушения существующих плагинов. Иногда опечатки становятся стандартами, возьмите к примеру HTTP-заголовок «referer»
3)
Lexer — часть парсера, анализирующая «сырую» (исходную) статью вики.
4)
Совместимые с Perl регулярные выражения. Ссылка: www.php.net/manual/en/ref.pcre.php.
5)
Криво переведено с англ.
6)
Существует ещё специальное состояние `metadata`, которое ничего не выводит, только собирает метаданные для страницы. Используйте его для вставки значений в массив метаданных.
7)
Название не может содержать символов подчерка и должно совпадать с названием класса