TypoScript **за** 45 **минут**

Ключ расширения: doc_tut_ts45

Язык: ru

Версия: 1.1.0

Ключевые слова: forAdmins, forBeginners, forIntermediates

Официальная документация

Этот документ является частью официальной документации TYPO3. Одобрение было получено после экспертной оценки командой по документации TYPO3. Читатель вправе ожидать точной информации в этом документе, о любом несоответствии сообщайте в команду по документации (documentation@typo3.org). Официальные документы сохраняются по возможности актуальными командой по документации.

Перевод официальной документации

Этот документ является переводом официальной документации TYPO3. Перевод может быть не столь актуальным, как оригинал. В случае вопросов, обращайтесь к английской версии.

Учебник

Этот документ является учебным руководством. Такие документы

разрабатываются в виде пошаговой инструкции, специально созданной практической задачи, выполняемой учеником от начала до конца. Для облегчения задачи, руководство снабжается примерами, иллюстрирующими изучаемый предмет. Кроме того, учебники дают представление о том, как избежать распространенных ошибок, выделяют ключевые понятия, которые пригодятся в дальнейшем.

Содержание

Введение

Об этом документе

Этот документ должен дать представление о работе TypoScript и о том, чем на самом деле он является. Он поможет разобраться в коде на самом деле, а не пользоваться слепым копированием-вставкой.

Нововведения

Этот документ является публикацией страницы wiki http://wiki.typo3.org/Ts45min. Все примеры должны работать со стабильным выпуском TYPO3 версии 4.4.

Составители

Оригинал этого руководства был создан в Германии, людьми, входящими в сообщество TYPO3 на wiki в wiki.typo3.org, а затем было переведено и исправлено остальными членами сообщества, за что им большое спасибо ;-)

Обратная связь

По общим вопросам о документации, пишите на documentation@typo3.org.

Если найдена ошибка в этом руководстве, опишите проблему по данному руководству в системе отслеживания ошибок: http://forge.typo3.org/projects/typo3v4-doc_tut_ts45/issues

Поддержка качественной документации является тяжелой работой, и команде по документации всегда нужны добровольцы. Если Вы желаете помочь, присоединяйтесь к списку рассылок (typo3.projects.documentation на lists.typo3.org).

TypoScript - краткий обзор

Введение

Целью введения является не то, чтобы в конце читатель сказал, - «это работает!», а чтобы он подумал, - «я все понял!». Другими словами, введение должно дать понимание работы TypoScript. Обычно кто-либо добавляет некие свойства к объекту, но знающий TypoScript сразу скажет, что данное свойство не сработает. Знание предмета сильно экономит время. Поиск и исправление ошибок упрощается для того, кто знает TypoScript, а иначе работа превращается в гадание на кофейной гуще.

Целью введения является не получение работающей установки TYPO3, а понимание, почему она работает.

Настройки внутреннего интерфейса

TypoScript влияет на многие части. Если TypoScript используется в поле TypoScript пользователя/группы пользователей, он меняет вид и поведение форм внутреннего интерфейса.

Формирование внешнего интерфейса определяется TypoScript в шаблоне TypoScript. Этот документ охватывает только лишь часть, относящуюся к формированию внешнего интерфейса и только посредством общего TypoScript.

Предварительные условия

Подразумевается, что читатель уже имеет настроенную и работающую систему TYPO3, и ему известны основные операции. Здесь не будет объясняться различие между страницами и элементами содержимого. И то, что содержимое страницы собирается из различных элементов содержимого, также должно быть известно Вам. И на всякий случай, мы скажем, что эти элементы содержимого хранятся в таблице tt_content. Поле базы данных "CType" определяет тип элемента содержимого. В зависимости от CType загружается определенная маска.

Для ясного понимания TYPO3 и TypoScript полезно посмотреть в базу данных. Расширение phpmyadmin позволяет просто и комфортно получить такой доступ из внутреннего интерфейса, таким образом можно получить представление о связи между страницами (pages), tt_content и внутреннем интерфейсом. Отсюда должно быть понятно, что PID (Page ID) используется для ID страницы, а UID (Unique ID) — для уникальной записи.

Почему TypoScript?

Строго говоря, TypoScript язык для настроек. Мы не можем при помощи него писать программы, но мы сможем с его помощь с легкостью настроить многое. Посредством TypoScript мы можем определить формирование сайта. Мы определяем навигацию, фиксированное содержимое, а также, как каждый элемент содержимого формируется на страницах.

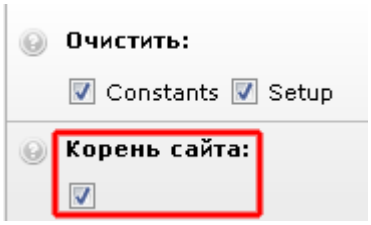
TYPO3 представляет из себя систему управления содержимым на сайте, целью которой является отделение содержимого от дизайна, внешнего вида. TypoScript можно рассматривать как клей, заново объединяющий упомянутые части (содержимое и его внешний вид). Содержимое, хранящееся в базе данных, читается и обрабатывается TypoScript, а затем отдается во внешний интерфейс.

Для формирования сайта, нам нужно только лишь определить **что** и **как** нужно передать.

- Ответ на вопрос "**что**" находится под контролем внутреннего интерфейса — здесь формируются страницы и их содержимое.
- Ответ на вопрос "**как**" находиться во власти TypoScript.

Посредством TypoScript мы определяем, как каждый из элементов содержимого будет представлен во внешнем интерфейсе. Например, в каком div-контейнере будет содержаться элемент, и будет ли заголовок обозначен, как <h1>

TypoScript, определяющий то, как будут отдаваться страницы, находится в "основном" шаблоне. В нем установлен флаг "корень сайта".



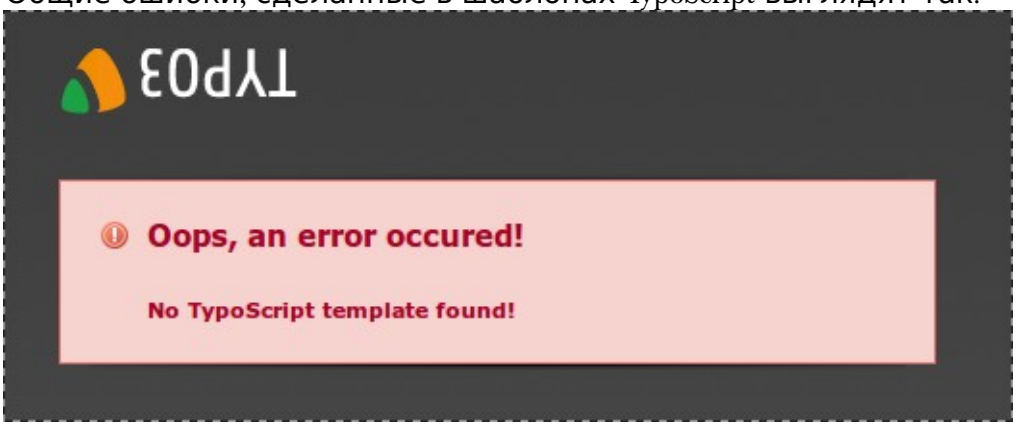
При отдаче страницы во внешний интерфейс, TYPO3 ищет по дереву страниц основной шаблон. Обычно, кроме основного, имеются несколько шаблонов. Как они совместно используются, хорошо видно в модуле Анализ шаблона. На данный момент мы подразумеваем, что шаблон один.

Синтаксис TypoScript очень прост. Слева находятся объекты и определяющие их свойства, которые получают соответствующие значения. Объект отделяется от свойства (которое тоже может содержать другой объект) точкой ".".

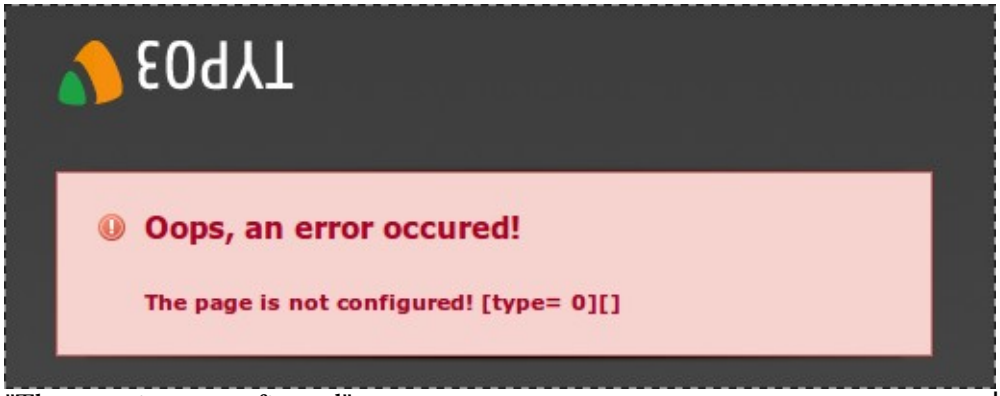
Термин «шаблон»

Шаблон имеет в TYPO3 двойное значение. С одной стороны — это шаблон HTML, служащий каркасом для всего выдаваемого содержимого. С другой стороны — это шаблон TypoScript, который может быть создан на страницах.

Общие ошибки, сделанные в шаблонах TypoScript выглядят так:



"No template found": предупреждение, появляющееся, если не найден шаблон, со включенным флагом "корень сайта".



"The page is not configured": это предупреждение появляется, если флаг "корень сайта" установлен, но не может быть найден объект PAGE.

Для исправления такой ошибки достаточно следующего кода:

```
page = PAGE
page.10 = TEXT
page.10.value = Hello World
```

TypoScript — это просто массив

TypoScript просто храниться во внутреннем массиве PHP. Он используется и оценивается различными классами в соответствии с типами объектов.

```
page = PAGE
page.10 = TEXT
page.10.value = Hello World
page.10.wrap = <h2>|</h2>
```

будет преобразован в следующий массив PHP:

```
$data['page'] = 'PAGE';
$data['page.'][10] = 'TEXT';
$data['page.']['10.']['value'] = 'Hello World';
$data['page.']['10.']['wrap'] = '<h2>|</h2>';
```

При оценке, сначала создается объект "PAGE", которому назначается параметр $data['page.']. Затем производится поиск всех свойств, определяющих объект "PAGE". В данном случае находится и оценивается лишь цифровая запись "10". Создается новый объект "TEXT" с параметром $data['page.']['10.']. Для объекта "TEXT" известен лишь параметр "value" с соответствующим содержимым. Все остальные параметры передаются в функцию stdWrap (о том, как обрабатывается "TEXT", мы поговорим позже, в связи со stdWrap). Здесь известно свойство 'wrap', и текст помещается на позицию вертикальной черты (|), а затем все выводиться.

Эти отношения важны для понимания поведения TypoScript с разных сторон. Если дополнить TypoScript, например следующей строкой:

```
page.10.myFunction = Magic!
```

элемент будет помещен в массив PHP:

```
$data['page.']['10.']['myFunction'] = 'Magic!';
```

Но ни объекту TEXT, ни функции stdWrap неизвестно свойство "myFunction". Соответственно, элемент не окажет никакого действия.

Никакой проверки на предмет ошибок в семантике не производится. Это особенно нужно учесть при поиске ошибок.

Первые шаги

Настройка основного шаблона определяет фундамент для выводимого на сайт.

TypoScript по существу состоит из объектов, имеющих определенные свойства. Некоторые из этих свойств могут восприниматься другими Объектами, другие нужны для определенных функций, либо определяют поведение объекта.

За вводимое на сайт отвечает объект PAGE.

```
# объект mypage задается как объект PAGE
mypage = PAGE

# у него есть свойство typeNum
mypage.typeNum = 0

# и объект "10" типа TEXT
mypage.10 = TEXT

# у объекта "10", в свою очередь, есть свойство "value"
mypage.10.value = Hello World
```

У объекта PAGE, в дополнение к многочисленным свойствам, имеется имеется бесконечное количество объектов, которые могут определятся только лишь номерами (так называемый массив содержимого). Это значит, что эти объекты обозначены лишь номерами, в соответствии с ними они и упорядочиваются при выводе на страницу. Первым идет объект с наименьшим номером, последним — с наибольшим. Порядок их появления в TypoScript не имеет значения.

```
mypage.30 = TEXT
mypage.30.value = This is last

# Первым будет выведен объект номер 10, затем 20 и 30. Объект номер 25 по логике
# будет выведен между объектами 20 и 30
mypage.20 = TEXT
mypage.20.value = I'm the middle

# Этот объект будет выведен первым
mypage.10 = TEXT
mypage.10.value = Hello World!

# здесь мы создаем второй объект для режима печати
print = PAGE
print.typeNum = 98
print.10 = TEXT
print.10.value = This is what the printer will see.
```

Каждый элемент хранится в многомерном массиве PHP. Поэтому, каждый Объект и каждое его свойство — уникальны. Мы можем определить случайный номер для объектов PAGE, но typeNum должен оставаться уникальным. Для каждого typeNum может существовать лишь один объект PAGE.

В приведенном примере, параметр typeNum = 98 создает другой режим вывода. Обычно для вывода HTML используется typeNum = 0. Запрос для вывода страницы на HTML будет index.php?id=1, соответственно index.php?id=1&type=98 служит запросом страницы для печати. Значение &type определяет отображаемый объект PAGE. Вот почему возможен режим печати, вывод HTML и даже вывод в виде PDF в одной и той же конфигурации. Для этого настройка, использующаяся для общего вывода может быть скопирована и немного изменена в новом объекте (например, обычное содержимое страницы можно скопировать в режим печати, но убрать оттуда меню).

Примечание

выводимое в этих примерах было обычным текстом. Но при выводе

форматов, вроде WML, заголовок (header) HTTP также должен быть изменен. Здесь это не затрагивается.

Предыдущий пример будет таким массивом PHP:

```
$TypoScript['mypage'] = 'PAGE';
$TypoScript['mypage .']['typeNum'] = 0;
$TypoScript['mypage .']['10'] = 'TEXT';
$TypoScript['mypage .']['10.']['value'] = 'Hello World!';
$TypoScript['mypage .']['20'] = 'TEXT';
$TypoScript['mypage .']['20.']['value'] = 'I'm the middle';
$TypoScript['mypage .']['30'] = 'TEXT';
$TypoScript['mypage .']['30.']['value'] = 'This is last';
```

Пробелы в начале и конце будут удалены в TYPO3 (trim()).

Знак "=" служит знаком соответствия: назначается значение.

```
# = установка значения
test = TEXT
test.value = Holla

# < Объект копируется
# mypage.10 returns "Holla"
mypage.10 < test

# Скопированный объект изменяется
# Изменения не отражаются на mypage.10
test.value = Hello world

# <= означает, что объект является ссылкой (объект связан)
test.value = Holla
mypage.10 <= test

# - Объект, являющийся ссылкой меняется
# - изменения ОТРАЖАЮТСЯ на mypage.10
# - mypage.10 вернет Hello world
test.value = Hello world
```

Объекты всегда записываются в верхнем регистре, для параметров и функций используется горбатаяЗапись (camelCase — первое слово с маленькой буквы, последующие начинаются с заглавной). Существует несколько исключений.

Посредством ".", служащей разделителем параметров, ссылаются на функции и дочерние объекты , которым могут быть назначены соответствующие значения.

```
mypage.10.wrap = <h1>|</h1>
```

Какие имеются объекты, параметры и функции, описано в TypoScript Справочник (TSRef).

Если некоторые объекты входят друг в друга, и назначается множество параметров, то это можно сократить.

```
mypage = PAGE
mypage.typeNum = 0
mypage.10 = TEXT
mypage.10.value = Hello world
mypage.10.typolink.parameter = http://www.typo3.org/
mypage.10.typolink.additionalParams = &nothing=nothing

# ATagParams к сожалению не использует "горбатуюЗапись"
mypage.10.typolink.ATagParams = class="externalwebsite"
mypage.10.typolink.extTarget = _blank
mypage.10.typolink.title = The website of TYPO3
mypage.10.postCObject = HTML
mypage.10.postCObject.value = This Text also appears in the link text
mypage.10.postCObject.value.wrap = |, because the postCobject is executed before the
typolink function
```

Для упрощения можно использовать фигурные скобки {}, определяющие

уровень объектов. Круглые скобки () - для написания текста в несколько строк, как в приведенном ниже примере:

```
mypage = PAGE
mypage {

  typeNum = 0

  10 = TEXT
  10 {

     value = Hello world
     typolink {

        parameter = http://www.typo3.org/
        additionalParams = &nothing=nothing

        # ATagParams ик сожалению не использует "горбатуюЗапись"
        ATagParams = class="externalwebsite"

        extTarget = _blank
        title = The website of TYPO3
     }

     postCObject = HTML
     postCObject {

        value = This Text also appears in the link text
        value {
           wrap (
            |, because the postCObject is executed before the typolink function
            )
         }
      }
   }
}
```

Опасность опечатки уменьшается, а сценарий лучше читается. Кроме того, если нам нужно будет переименовать mypage, то изменить нужно лишь две строки, вместо всего сценария.

Чтение записей содержимого

Примечание

следующие параграфы служат примером, для лучшего понимания лежащий в основе процессов и связей. Следующие сценарии взяты из css_styled_content, поэтому их необязательно писать вручную. Если нужно в корне изменить вид элемента содержимого, либо при программировании расширения с новым элементом содержимого, то необходимо понять связи.

Мы не хотим вводить все содержимое через TypoScript —это утомительное занятие и мы не можем ожидать этого от редакторов.

Поэтому мы создадим элемент содержимого, типа "TEXT" и TypoScript, автоматически собирающий содержимое. В этом примере создается страница с заголовком и текстом из всех элементов содержимого на текущей странице.

Для начала создаем объект PAGE, чтобы что-то могло быть выведено. В этом объекте PAGE мы создаем объект CONTENT, которым можно управлять при помощи различных параметров TypoScript.

```
page = PAGE
page.typeNum = 0

# Объект content выполняет запросы к базе данных и загружает содержимое
page.10 = CONTENT
page.10.table = tt_content
page.10.select {

     # "sorting"- это столбец из таблицы tt_content, где
     # отслеживается порядок, заданный во внутреннем интерфейсе
     orderBy = sorting

     # столбец normal
     where = colPos = 0
}

#  renderObj выполняется для каждой строки-результата из запроса к базе данных,
# а внутренний массив данных заполняется содержимым. Это гарантирует нам
# возможность вызова свойства .field и мы получаем соответствующее значение
page.10.renderObj = COA
page.10.renderObj {

  10 = TEXT

  # Поле tt_content.header обычно содержит заголовок.
  10.field = header

  10.wrap = <h1>|</h1>

  20 = TEXT

  # Поле tt_content.bodytext содержит текст содержимого
  20.field = bodytext

  20.wrap = <p>|</p>
}
```

Объект CONTENT выполняет запрос SQL к базе данных. За запрос отвечает "select". "Select" определяет, что нам нужны все записи из столбца 0 (являющийся столбцом "NORMAL" во внутреннем интерфейсе), а порядок выдачи результата определяется полем "sorting". Если свойство pidInList не установлено или удалено, запрос будет ограничен текущей страницей. Например, если страница с ID 100 ссылается на объект CONTENT, то будут возвращены лишь записи со страницы с pid=100.

Свойство renderObj определяет то, как записи будут выданы. Здесь оно было определено, как COA (Content Object Array — массив объектов содержимого), который может содержать любое количество объектов TypoScript. В данном случае используются два объекта TEXT, выводимые друг за другом. Порядок их вывода не зависит от порядка следования в коде TypoScript, а присвоенным им номером. Объект TEXT "10" выводится перед объектом TEXT "20".

Недостаток в том, что все элементы содержимого типа текста будут выведены, как их предопределил веб дизайнер. Поэтому нам нужно создать определение для каждого поля (например для изображений, размер изображений, местоположение, индекс и т.п.).

Различные элементы содержимого

Если вместо текста нам нужно вывести изображение, то нужно выбрать другое поле из tt_content и вывести его не так, как обычный текст . То же применимо к "тексту с изображением", "заголовку" и т.п.

Тип элемента содержимого хранится в столбце tt_content.CType. В следующем примере мы покажем, как при помощи объекта CASE можно разделить вид каждого элемента содержимого.

```
10.renderObj = CASE
10.renderObj {

  # для разделения будет использоваться поле CType
  key.field = CType

  # Тип содержимого "headline" хранится внутри, как "header"
  header = TEXT
  header.field = header
  header.wrap = <h1>|</h1>

  # Текст используется для текста элемента содержимого
  text = COA
  text {

    10 = TEXT
    # Поле tt_content.header обычно содержит заголовок.
    10.field = header
    10.wrap = <h1>|</h1>

    20 = TEXT
    # Поле tt_content.bodytext содержит текст содержимого
    20.field = bodytext
    20.wrap = <p>|</p>

  }

  # ... другие определения
}
```

css_styled_content

Утомительно программировать одно и тоже при каждой установке TYPO3, так как элементы одинаковые и имеют схожие функции. По этим причинам в TYPO3 появились "статические шаблоны". Текущая версия использует "css_styled_content". Это удобное определение каждого из существующих элементов содержимого.

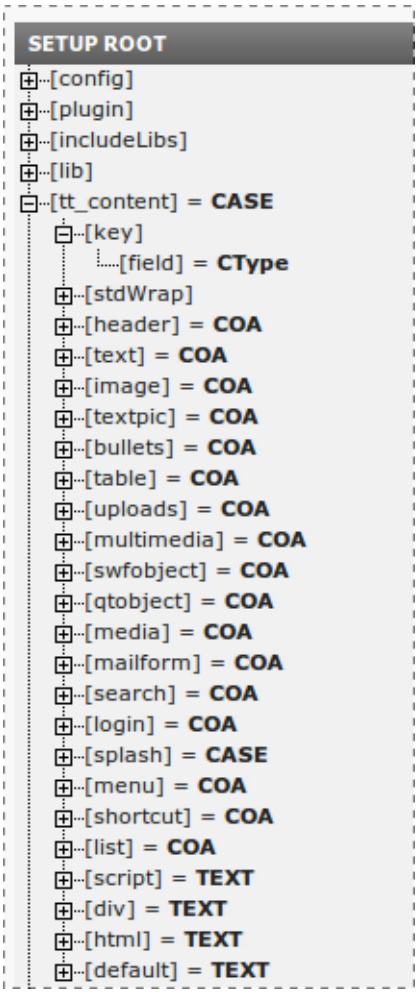
Использование очень простое. Определение доступно в качестве объектов tt_content.

```
10.renderObj < tt_content
```

Это назначение — настройка по умолчанию для элемента CONTENT. Если доступен статический шаблон "css_styled_content", то нет нужды использовать

параметр "renderObj".

Таким образом, для каждого элемента содержимого в TYPO3 существует соответствующее определение в css_styles_content. В проводнике объектов это выглядит примерно так:



Таким образом становиться понятно, как настроен каждый из элементов содержимого. Если нужно настроить элемент содержимого в корне отличным способом, должно быть ясно, что сделать это можно через tt_content."внутренний идентификатор элемента содержимого". Вот пример того, как можно переписать стандартное свойство для заголовка (header):

```
# Так как TYPO3 все хранит в одном большом массиве, не переназначаемые свойства сохраняются
# и привести к неожиданным результатам. Вот почему старые свойства нужно удалить полностью.
tt_content.header >

# Каждый заголовок будет выведен, как H1, независимо от свойств элемента содержимого.
tt_content.header = TEXT
tt_content.header.wrap = <h1>|</h1>
tt_content.header.field = header
```

Поэтому не нужно воссоздавать не только "renderObj", но и объект CONTENT уже определенный в css_styled_content.

styles.content.get

```
# наш код до сих пор
page.10 = CONTENT
page.10.table = tt_content
page.10.select {

    # Вместо сортировки из внутреннего интерфейса, можно использовать также поле date или
header
    orderBy = sorting

    # столбец normal
    where = colPos = 0
}
```

Благодаря css_styled_content, для достижения того же результата достаточно следующей записи:

```
# Возвращение содержимого из столбца "normal" (colPos = 0)
page.10 < styles.content.get
```

Для других столбцов также есть определения по умолчанию:

```
# Возвращение содержимого из столбца "left" (colPos = 1)
page.10 < styles.content.getLeft

# Возвращение содержимого из столбца "right" (colPos = 2)
page.10 < styles.content.getRight

# Возвращение содержимого из столбца "border" (colPos = 3)
page.10 < styles.content.getBorder
```

В css_styled_content, border определяется следующим образом:

```
# Копируется столбец normal
styles.content.getBorder < styles.content.get

# после этого изменяется colPos
styles.content.getBorder.select.where = colPos=3
```

Создание меню

До сих пор мы изучали формирование содержимого страницы, опуская из вида навигацию.

TYPO3 предоставляет специальный объект меню, HMENU (H значит hierarchic — иерархическое) для упрощения создания различных видов меню.

Меню должно строиться в виде вложенного списка:

```
<ul>
   <li>первый уровень</li>
   <li>первый уровень
       <ul>
           <li>второй уровень</li>
       </ul>
   </li>
   <li>первый уровень</li>
</ul>
```

Для сохранения порядка, мы создаем новую папку и дополнительный шаблон в ней. Здесь мы определяем новый объект, который позже можем добавить в основной шаблон. Таким образом мы можем создавать множество объектов по отдельности и сохранять их для будущих проектов. Дополнительные шаблоны можно добавить в основной, используя поле "подключить базовые шаблоны".

Обычно такие объекты определяются как потомки объекта "lib". Вообще мы можем использовать любое, еще не использующееся слово.

```
lib.textmenu = HMENU
lib.textmenu {

  # первый уровень определяется как текстовое меню
  1 = TMENU

  # Определяем режим ''NO''rmal - обычный
  1.NO.allWrap = <li>|</li>

  # Определяем режим ''ACT''ive - активный
  1.ACT = 1
  1.ACT.wrapItemAndSub = <li>|</li>

  # Обертка для всего уровня меню
  1.wrap = <ul class="level1">|</ul>

  # Второй уровень настраивается схожим образом.
  # Можно воспользоваться копией уже определенных внутри скобок объектов.
  # Точкой "." мы определяем, что объект можно найти в скобках
  2 < .1
  2.wrap = <ul class="level2">|</ul>
  3 < .1
  3.wrap = <ul class="level3">|</ul>
}
```

Объект HMENU позволяет создавать разнообразные меню. Для каждого уровня меню может быть создан свой объект меню, выводящий содержимое. Так можно создавать GMENU на первом уровне, а для второго и последующих использовать TMENU.

Первый уровень меню определяется цифрой 1, второй — 2 и т.д. Нумерацию нельзя пропускать (например, если третий уровень не определен, четвертый не будет сформирован).

На каждом уровне меню мы можем настроить различные состояния его элементов (NO = "обычное" ACT = "страница в корневой линии, значит текущая страница, родитель, прародитель и т.п.", CUR = "текущая (CURrent) страница"). При этом, обратите особое внимание, что, за исключением обычного состояния ("NO"), каждое состояние сначала должно быть активировано (например, ACT = 1).

После этого мы можем использовать это меню, применяя его для нашей страницы так:

```
page.5 < lib.textmenu
```

Помещение содержимого в шаблон

Теперь нам известно, как формировать содержимое и как создавать меню, но у нас все еще нет настоящего сайта.

Мы можем создавать веб сайт посредством COA, а каркас HTML через TypoScript Но это было бы очень сложным и с большой возможностью допустить ошибку. Если шаблон HTML был создан дизайнером и его разработка закончена, то все еще более усложняется, особенно при дальнейшем внесении поправок в шаблон.

Поэтому имеется элемент TEMPLATE, посредством которого мы можем анализировать шаблон HTML и вставлять в его определенные места меню, содержимое и т.п.

```
page.10 = TEMPLATE
page.10 {
 template = FILE

 # Загрузка шаблона HTML
 template.file = fileadmin/test.tmpl

 # Область текста
 # <!-- ###MENU### begin -->
 # Пример указателя содержимого, все между маркерами будет
 # заменено на содержимое подразделов, в данном случае - на меню
 # <!-- ###MENU### end -->

 subparts {
   MENU < lib.textmenu
   INHALT < styles.content.get
   SPALTERECHTS < styles.content.getRight
 }

 # Разметка отдельных маркеров, например здесь нет маркеров начала и окончания,
 # вместо этого заменяется непосредственно сам маркер. ###LOGO### будет
 # заменено на логотип.
 marks {
   LOGO = IMAGE

   # Рисунок logo*.gif должен быть добавлен в поле ресурсов шаблона TypoScript
   LOGO.file = logo*.gif

   # Ссылка с логотипа ведет на страницу с ID 1
   LOGO.stdWrap.typolink.parameter = 1
 }
 workOnSubpart = DOCUMENT
}
```

Альтернативой такому решению может послужить расширение automaketemplate, с помощью которого можно полностью обработать маркеры. Вместо использования ID в качестве ссылок, можно лучше настроить взаимодействие с дизайнером шаблона.

Другой альтернативой служит расширение templavoila. В нем используется визуальный пользовательский интерфейс. Но оно не рекомендуется для новичков.

Использование css_styled_content

Мы уже увидели, что можем и сами определить различные элементы содержимого в Typo3. Но css_styled_content уменьшает объем работы примерно на 2000 строк TypoScript.

Полезно, даже если в начале ничего не понятно, уметь разбираться в TypoScript из TYPO3. Для этого нужно перейти на нужную для анализа страницу, имеющую шаблон. Затем в модуле "Шаблон" нужно выбрать "Анализ шаблона" в меню сверху.

Появится список активных и интегрированных шаблонов TypoScript. Они анализируются TYPO3 снизу вверх и объединяются в один массив настроек.

Щелкнув по "EXT:css_styled_content/static/" можно вывести содержимое этого шаблона. Сначала выводятся константы, а затем настройки (setup) TypoScript.

Расширение css_styled_content добавляет множество классов в элементы HTML. Преимущество в том, что не нужно вводить все классы вручную. Достаточно найти элемент HTML с определенным классом и добавить к нему стиль CSS.

Пример:

```
<div class="csc-textpic-imagewrap">...
```

Описания классов простое и, если знакомы с внутренним устройством TYPO3, интуитивно понятны. Все классы начинаются с "csc", это значит "css_styled_content". В примере далее следует "textpic", что означает TypoScript элемент "textpic" (текст с изображением). Кроме того, используется "imagewrap" — значит картинка помещена в контейнер div.

Детально изучит происходящее можно создав пустую страницу с одним элементом, а затем просмотреть сформированный код страницы.

Например, заголовки обычно следуют так, чтобы первый мог быть специфично обработан. Для таблиц HTML вставляются классы "odd" и "even" для упрощения чередования расцветки рядов таблицы. Таким же образом можно обработать и столбцы таблиц.

Для пуристов HTML это означает использование слишком многочисленных классов css, что недопустимо. Чтобы избавиться от них, нужно потрудиться над редактированием расширения css_styled_content.

COA объекты TypoScript

Объекты TypoScript применяются в TYPO3 по соответствующим классам. При различных условиях на веб страницу выводятся различные объекты. У этих объектов есть набор свойств. Например, объект IMAGE имеет метод wrap и метод titleText. В справочнике по TypoScript можно найти, какие типы значений поддерживает тот или иной объект. Для wrap поддерживается тип данных wrap — текст, разделенный вертикальной чертой (|). Поэтому здесь бесполезно использовать некоторые функции (например "wrap.crop = 100").

Объект получает параметры (как было сказано выше) в массиве PHP (например, $conf['wrap.']['crop']='100';), он может включать любое количество записей. Но будут использованы только те, на которые ссылается объект (например, $conf['wrap'] или $conf['titleText']).

В случае "titleText" типом данных является "string / stdWrap", это значит, что допустимы как текст(string), так и метод типа stdWrap. Какие свойства допускает stdWrap, можно найти в справке по stdWrap. Следовательно, мы можем расширить метод "titleText" посредством различных свойств из stdWrap (например: titleText.field = header). При этом сначала значением titleText будет служит обычный текст, а затем к нему будут применены функции stdWrap.

Поэтому нет надобности предполагать, что можно делать с объектом, нужно просто поискать эту информацию в справочнике.

На веб страницу выводится множество объектов. Фокус в том, как их правильно объединить.

В разделе "Чтение записей содержимого" мы увидели, как можно использовать объект CONTENT для выполнения запроса к базе данных и возвращать содержимое страницы. Объект получает список элементов содержимого, созданных друг за другом (обычно в порядке сортировки). Поэтому, для разбивки типа элементов (CType) мы использовали объект CASE и выводили их по разному.

Крайне важно знать различные объекты и функции TypoScript.

Объекты, выполняющие запросы к базе данных

- CONTENT предоставляет функционал для доступа к различным таблицам внутри TYPO3. Это не только таблица tt_content, но и, например, таблицы расширений. Функция select позволяет выполнять сложные запросы SQL.
- RECORDS дает возможность ссылаться на определенные записи данных. Очень полезно выводить одинаковый текст на всех страницах. Используя RECORDS, можно определить показываемый элемент содержимого. При этом редактор может отредактировать его содержимое однажды, а не на каждой странице. Объект также используется при применении элемента содержимого "вставить запись".
  В следующем примере, адрес email записи адреса обрабатывается и оформляется в виде ссылки на e-mail.

```
page.80 = RECORDS
page.80 {
      source = 1
      tables = tt_address
      conf.tt_address = COA
      conf.tt_address {
              20 = TEXT
              20.field = email
```

```
                20.typolink.parameter.field = email
        }
}
```

- HMENU импортирует для дерева страниц и дает комфортный способ формирования меню страниц. Кроме меню, анализирующего дерево страниц, имеется несколько специальных меню для специфического применения. Этот объект импортирует внутреннюю структуру для меню. А то, как эти меню формируются, зависит от объектов меню, вроде TMENU (текстовое меню) или GMENU (графическое меню). Для каждого уровня меню, объект может быть изменен. Внутри каждого уровня меню имеются различные элементы меню. Для каждого из них можно определить различные состояния (NO = обычное, ACT = активное и т.д.).

Объекты для формирования содержимого

- IMAGE отвечает за формирование изображений

```
lib.logo = IMAGE
lib.logo {
  file = fileadmin/logo.gif
  file.width = 200
  stdWrap.typolink.parameter = 1
}
```

lib.logo содержит логотип, шириной 200 пикселей, который имеет ссылку на страницу с PID 1.

- HTML / TEXT служат для обработки обычного текста или полей содержимого. Главное отличие: объект HTML применяет функционал stdWrap в свойстве .value

```
lib.test1 = TEXT
lib.test1.field = uid

lib.test2 = HTML
lib.test2.value.field = uid
```

- FILE отвечает за непосредственный импорт содержимого файла.
- TEMPLATE заменяет маркеры в шаблоне на содержимое:

```
page.10 = TEMPLATE
page.10 {
  template = FILE
  template.file = fileadmin/test.tmpl
  subparts {
    HELLO = TEXT
    HELLO.value = r заменяет содержимое между маркерами ###HELLO### и ###HELLO###
  }
  marks {
    Test = TEXT
    Test.value = маркер Test будет заменен этим текстом
  }
  workOnSubpart = DOCUMENT
}
```

- MULTIMEDIA обрабатывает мультимедиа объекты.
- IMGTEXT позволяет формировать встроенные в текст изображения. Используется для элемента содержимого "текст с изображением".
- FORM генерирует формы HTML.

Другие объекты

- CASE этот объект выполняет разделение по условиям. В css_styled_content этот объект используется для формирования разных объектов в соответствии с их типом.

- COA - Content Object Array - массив объектов содержимого, позволяет комбинировать любое количество объектов.
- COA_INT - не кешируемый. Этот элемент будет формироваться заново при каждом вызове. Полезно, например, для времени/даты или данных, зависящих от пользователя.
- LOAD_REGISTER / RESTORE_REGISTER при помощи этого объекта можно заносить содержимое в глобальный массив $GLOBALS["TSFE"]->register[]. Объект ничего не возвращает. Можно использовать как единичные значения, так и объекты TypoScript . Регистр работает наподобие стека — при каждом вызове заноситься новый элемент. Через RESTORE_REGISTER элемент сверху можно удалить.
- USER и USER_INT определяемые пользователем функции, каждое дополнение является таким объектом. USER_INT — не кешируемый вариант.
- IMG_RESOURCE используется в IMAGE. Возвращается ресурс, содержимое, обычно присутствующее в атрибуте SRC тега IMG. При масштабировании изображений, этот объект служит основой для нового файла, сохраняемого в папке /typo3temp.
- EDITPANEL - этот объект вставляется, если авторизован внутренний пользователь и в панели внешнего пользователя установлен параметр "Отображать значки редактирования". Страницы не кешируются, отображаются значки для редактирования: упорядочивание, правка, удаление и т.п.
- GIFBUILDER используется для динамического формирования файлов GIF. Можно комбинировать текст с изображением и т.п. Сам по себе GIFBUILDER предлагает некоторые объекты, вроде TEXT или IMAGE, не относящиеся к обычным объектам TEXT и IMAGE соответственно. Работая с GIFBUILDER нужно помнить об этом и не путать объекты, носящие одинаковое наименование.

Мы не затронули все объекты, существующие в TypoScript, но упомянули наиболее важные из них.

Функции TypoScript

Функции TypoScript используются для изменения и настройки выводимого элементами содержимого. Наиболее популярной является функция стандартной обертки, известная как stdWrap. Применима ли к объекту какая-либо из функций, показано в TSRef, в колонке тип данных.

Свойство	Тип данных	Описание	По умолчанию
file	imgResource		
imageLinkWrap	->imageLinkWrap	[...]	
if	->if	[...]	
altText **titleText**	String/stdWrap	[...]	

[Пример:(cObject).IMAGE]

Первая строка в этом примере (свойство = file) говорит, что свойства файл (file) типом данных является imgResource. Это значит, что мы можем использовать функции imgResource в свойстве file.

Иногда функции, для лучшего распознавания, помечаются стрелочкой (вроде -> if).

Если несколько типов разделены слешем, то можно по разному использовать этот элемент. В приведенном примере это видно для title- и altText. Оба могут быть либо обычной строкой, либо stdWrap. Можно присвоить обычную строку и ничего кроме, либо настроить и изменить ее, используя возможности stdWrap, либо оставить строку пустой, а содержимое формировать только лишь через stdWrap.

Наиболее важные и часто используемые функции представлены в следующих подразделах. Эта глава представляет эти функции и говорит об их применении. Детальная информация приведена в TSRef, а не здесь.

imgResource

Функции типа данных "imgResource" относятся, как видно из названия, к изменениям рисунков. Объект IMAGE имеет свойство "file" унаследованное из типа данных "imgResource".

Так, например, можно сделать изображение с изменяемыми размерами:

```
temp.myImage = IMAGE
temp.myImage {

        file = toplogo.gif
        file.width = 200
        file.height = 300

}
```

Ввод максимальных (или минимальных) размеров:

```
temp.myImage = IMAGE
temp.myImage {

        file = toplogo.gif

        # максимальное значение
        file.maxW = 200
        file.maxH = 300

        # минимальное значение
        file.minW = 100
```

```
        file.minH = 120

}
```

и прямой доступ к функции ImageMagick:

```
temp.myImage = IMAGE
temp.myImage {

        file = toplogo.gif
        file.params = -rotate 90
```

Одним из лучших и общих примеров использования imgResource является динамическое применение рисунков из поля Медиа свойств страницы. Таким образом редакторы в состоянии изменять изображение не используя TypoScript Это позволяет использовать разные изображения из свойств страницы в различных областях веб сайта посредством нескольких строк TypoScript:

```
temp.dynamicHeader = IMAGE
temp.dynamicHeader {
        file {

                # Определение пути к изображениям
                import = uploads/media/

                import {

                        # Если на текущей странице нет изображения, ищем выше по дереву
страниц
                        data = level:-1, slide

                        # Определение поля с изображением
                        field = media

                        # определение номера отображаемого изображения
                        # (в данном случае используется первое)
                        listNum = 0

                }
}
```

Путь "uploads/media/" является местоположением файлов, вставляемых на страницу через свойство "файлы" (для версий TYPO3 4.2.x оно расположено на вкладке "Ресурсы"). TypoScript в скобках import полностью состоит из функций stdWrap, использующихся для определения откуда и какое брать изображение. В итоге stdWrap возвращает название файла изображения, которое затем будет импортировано из местоположения (uploads/media).

imageLinkWrap

Используя "imageLinkWrap" можно создавать ссылку с изображения на сценарий PHP "showpic.php". Сценарий откроет изображение в новом окне с предопределенными параметрами, вроде фона, размера изображения и т.п. Можно использовать эту функцию для создания функциональности "увеличения по щелчку" (щелчок по маленькому изображению-эскизу открывает окно с полноразмерным изображением).

```
temp.meinBild = IMAGE
temp.meinBild {

        file = toplogo.gif

        imageLinkWrap = 1
        imageLinkWrap {

                # включение ImageLinkWrap
                enable = 1

                # определение тега body для нового окна
                bodyTag = <body class="BildOriginal">
```

```
                # обертка для изображения (закрытие окна по щелчку на изображении)
                wrap = <a href="javascript:close();"> | </a>

                # Ширина изображения (m позволяет соблюдать пропорции)
                width = 800m

                # высота изображения
                height = 600

                # создание нового окна для изображения
                JSwindow = 1

                # открывать новое окно для каждого из изображений
                # (иначе будет использовать одно и то же окно)
                JSwindow.newWindow = 1

                # Отступы для нового окна
                JSwindow.expand = 17,20
        }

}
```

numRows

В TypoScript имеются не только большие мощные функции, но и не менее мощные маленькие. Например, функция numRows, единственное предназначение которой — возвращать количество строк в запросе выборки. Точно как объект CONTENT, numRows использует функцию select. Запрос в обоих классах формируется схожим образом, разница в возвращаемом результате — количество строк или само содержимое этих строк.

Используя функцию "if", можно сформировать некоторые удобные кусочки кода. Например таблицу стилей для содержимого правого столбца внутреннего интерфейса, используемого только если в нем имеется некое фактическое содержимое.

```
temp.headerdata = TEXT
temp.headerdata {
        value = <link rel="stylesheet" type="text/css"
href="fileadmin/templates/rechteSpalte.css">

        # если select возвращает хотя бы 1 строку, вставляется таблица стилей
        if.isTrue.numRows {

                # проверка, является ли это страницей
                pidInList = this

                # имеется ли содержимое в таблице tt_content
                table = tt_content

                # SQL: WHERE colPos = 2
                select.where = colPos=2
        }
}

# копирование temp.headerdata в page.headerData.66 (и переназначение page.headerData.66)
page.headerData.66 < temp.headerdata
```

либо использование другого шаблона, если в правом столбце имеется содержимое:

```
# COA (Content Object Array) позволяет объединять несколько объектов
temp.maintemplate= COA
temp.maintemplate {

        # 10 будет встроено, только если if вернет "true"
        10 = COA
        10 {
                # используем копию select из css_styled_content
                if.isTrue.numRows < styles.content.getRight
```

```
                10 = TEMPLATE
                10 {
                        template = FILE
                        template.file = fileadmin/templates/template-2column.html
                }
        }

        # 20 будет встроено, только если if вернет "true"
        20 = COA
        20 {
                if.isFalse.numRows < styles.content.getRight
                10 = TEMPLATE
                10 {
                        template = FILE
                        template.file = fileadmin/templates/template.html
                }
        }
}
```

select

Функция "select" формирует запрос SQL SELECT, использующийся для чтения записей из базы данных. Функция select автоматически проверяет скрытые, удаленные и со сроком начала-окончания записи. Если используется pidInList (обозначение списка предоставляемых страниц), функция делает проверку на то, имеет ли текущий пользователь право на просмотр всех записей.

При помощи функции select, например, возможно показать содержимое определенной страницы на всех страницах.

```
temp.leftContent = CONTENT
temp.leftContent {

    table = tt_content
    select {

        # источником служит страница с ID = 123
        pidInList = 123

        # используется сортировка, указанная во внутреннем интерфейсе
        orderBy = sorting

        # содержимое столбца left
        where = colPos=1

        # определение поля с lanuguage-ID в tt_content
        languageField = sys_language_uid
    }
}

# копирование temp.leftContent в  marks.LEFT
marks.LEFT < temp.leftContent
```

split

Функция split полезна для разделения данных по предопределенному символу и последующей обработке частей. При каждой иттерации, сохраняется текущий ключ-индекс "SPLIT-COUNT" (начиная с 0).

Используя "split" можно, например, прочитать поле таблицы и заключить каждую отдельную строку в определенный код (например, сформировать таблицу HTML, отображающую одно и то же содержимое на нескольких страницах):

```
# Пример
20 = TEXT

# Импортируется содержимое поля "bodytext" (из $cObj->data-array)
```

```
20.field = bodytext
20.split {

  # Определение символа-разделителя (char = 10 – это перенос строки).
  token.char = 10

  # Определение нужного элемента.
  # Используя optionSplit мы можем различать элементы.
  # Должен быть определен элемент с соответствующим номером!
  # Здесь используется свойство option split.
  # Для представления будут использоваться числа 1 и 2 по очереди.
  # В этом примере используются классы "odd" и "even"
  # для возможности стилизации таблицы по типу зебры.
  cObjNum = 1 || 2

  # Первый элемент определен (ссылка в cObjNum)
  # Содержимое импортируется посредством stdWrap->current
  1.current = 1

  # Создается обертка для элемента
  1.wrap = <TR class="odd"><TD valign="top"> | </TD></TR>

  # Определяется второй элемент и его обертка
  2.current = 1
  2.wrap = <TR class="even"><TD valign="top"> | </TD></TR>
}

# Общая обертка, для создания правильно размеченной таблицы
20.wrap = <TABLE border="0" cellpadding="0" cellspacing="3" width="368"> | </TABLE>
```

if

Возможно самая сложная для понимания функция TYPO3 — это "if", так как каждый программист, инстинктивно использующий ее, делает это неправильно. Поэтому необходимо привести несколько примеров ее работы.

В общем, функция if возвращает "true" (истина), если все условия выполняются — это сродни булевой комбинации И. Если должно быть возвращено "false", можно использовать параметр "negate" для обращения результата (!(true)).

```
10 = TEXT
10 {

    # Содержимое текстового элемента
    value = The L parameter is passed as GET var

    # Если результат "true", формируется верхнее значение, то есть
    # GET/POST параметр передан со значением, не являющимся 0
    if.isTrue.data = GP:L
}
```

Используя "if", возможно сравнение значений. Для этого мы используем параметр if.value.

```
10 = TEXT
10 {

    # ВНИМАНИЕ: это значение схоже с текстовым выражением, а не с "if"
    value = 3 is bigger than 2

    # параметр для сравнения в функции "if"
    if.value = 2

    # Запомните: сравнение нужно читать с конца, это условие
    # эквивалентно предложению "3 isGreaterThan 2", что верно
    if.isGreaterThan = 3
}
```

Так как к свойствам функции "if" применимы функции stdWrap, могут сравниваться все виды переменных.

```
10 = TEXT
10 {
    # значение текстового элемента
```

```
        value = The record can be shown because the starting date has passed.

        # условие для if
        if.value.data = date:U

        # условие снова нужно читать с конца: start time isLessThan date:U
        if.isLessThan.field = starttime
    }
```

typolink

Typolink — это функция TYPO3, позволяющая формировать все виды ссылок. По возможности для формирования ссылок нужно использовать эту функцию, так как они "регистрируются" в TYPO3 —что является обязательным условием правильного функционирования, например realURL, формирующего правильные с точки зрения поисковых механизмов пути, или анти спам защиты адресов email. Если вы настойчиво используйте <a href="..."> —**откажитесь от этого**.

Применять typolink очень просто. Typolink оформляет текст в соответствии с установленными параметрами. Пример:

```
temp.link = TEXT
temp.link {

    # это определение текста
    value = Examplelink

    # далее следует функция typolink
    typolink {

        # что является целью ссылки?
        parameter = http://www.example.com/

        # где будет открыта ссылка(_blank открывает новое окно)
        extTarget = _blank

        # и добавление класса к ссылке, позволяющего стилизовать ее.
        ATagParams = class="linkclass"
    }
}
```

Приведенный пример сформирует следующий код HTML: Examplelink

Typolink схожа с wrap — содержимое, определяемое, например свойством value помещается в HTML тег ссылки. Если содержимого не определено, оно формируется автоматически. В ссылке на страницу, будет использован ее заголовок, в во внешней ссылке, будет показан ее URL.

Можно сократить этот пример, так как тег "parameter" из функции typolink додумает остальное за вас. Вот сокращенный пример, формирующий тот же код HTML.

```
temp.link2 = TEXT
temp.link2 {

    # снова определяем текст
    value = Examplelink

    # параметр является суммой параметров из первого примера (объяснение следует)
    typolink.parameter = www.example.com _blank linkclass
}
```

"parameter" является частью функции typolink, анализирующей значение на предмет специальных символов и производящей соответствующие преобразования. В начале значение разбивается на части по пробелам. Если в первой части имеется точка "." (в частности перед первым слешем "/"), будет сформирована внешняя ссылка. Если точна "." находится после слеша "/",

формируется ссылка на файл. Если присутствует @, формируется ссылка на e-mail. Если присутствует целое число, вроде "51", формируется внутренняя ссылка на страницу с id "51". Если в начале стоит решетка "#", будет установлена ссылка на определенный элемент содержимого (например, для ссылки на элемент содержимого с id #234 на текущей странице. Для ссылки на элемент #234 на странице с ID 51 нужно использовать 51#234).

Вторая часть в свойстве parameter описывает цель для ссылки. Обычно это, как в первом примере, определяется через extTarget (для внешних ссылок) или target (для внутренних), но может быть переназначено через второй параметр.

Третья часть будет преобразована в атрибут class для ссылки.

Если необходим только лишь атрибут class, без цели, то часть для цели необходимо заполнить все равно, так как функция не сможет распознать class, который должен находиться на третьем месте. Поэтому, без цели, строка будет следующей (в качестве разделителя используется знак "-")

```
typolink.parameter = www.example.com - linkclass
```

Используя функцию typolink и атрибут target, возможно открывать ссылки во всплывающем окне через JavaScript.

```
temp.link = TEXT
temp.link {

     # текст ссылки
     value = Open a popup window

     typolink {
          # 1. Parameter = PageID целевой страницы, 1 2. parameter = размер окна
          parameter = 10 500x400

          # Тег title для ссылки
          title = Click here to open a popup window

          # Параметры всплывающего окна
          JSwindow_params = menubar=0, scrollbars=0, toolbar=0, resizable=1

     }
}
```

Важно помнить, что многие свойства typolink имеют тип stdWrap. Это значит, что значения можно высчитать или взять из базы данных.

```
lib.stdheader >
lib.stdheader = TEXT
lib.stdheader {
   field = header
   typolink.parameter.field = header_link
   wrap = <h2>|</h2>
}
```

Будет показан заголовок, со ссылкой, имеющей цель, определенную в поле header_link. Первой строкой удаляются настройки по умолчанию из css_styled_content.

encapsLines

EncapsLines сокращение от "encapsulate lines" — инкапсуляция строк. Эта функция TypoScript позволяет определить, во что будут помещаться (какие теги) строки содержимого . Например, если ничего не определено, элемент помещается в <p> или . Другой пример — автоматическая замена всех тегов <b> на теги .

Простой пример:

В RTE имеется следующий текст:

```
Простой текст, без чего-либо.
```

```
<div class="myclass">Текст, заключенный в тег div.</div>
```

В TypoScript определено следующее:

```
encapsLines {
        # определение тегов, которые будут видны, как обертки
        encapsTagList = div,p

        # Строки, еще ни во что не заключенные из списка тегов
        # encapsTagList будут помещены в теги <p>
        wrapNonWrappedLines = <p>|</p>

        # замена всех тегов DIV на теги P
        remapTag.DIV = P

        # если строка пустая, вставить неразрывный пробел
        innerStdWrap_all.ifEmpty =  
}
```

В результате появится следующий код HTML на сайте:

```
<p>Простой текст, без чего-либо.</p>
<p> </p>
<p class="myclass">Текст, заключенный в тег div.</p>;
```

В большинстве проектов TYPO3 следующий код не понадобится. Но в расширении "css_styled_content" некоторые настройки сделаны через эту функцию и могут быть переопределены при необходимости. Далее следуем примеру стандартных настроек из "css_styled_content" для подстройки определенных функций.

```
lib.parseFunc_RTE {

        nonTypoTagStdWrap.encapsLines {

                # теги-обертки
                encapsTagList = div,p,pre,h1,h2,h3,h4,h5,h6

                # замена всех тегов DIV на теги P
                remapTag.DIV = P

                # обертка всех не обернутых строк в тег <p>
                nonWrappedTag = P

                # замена пустых строк на неразрывный пробел
                 innerStdWrap_all.ifBlank =  

                # здесь помещается ненужный bodytext
                addAttributes.P.class = bodytext

                #  использование addAttributes если не установлено никаких атрибутов
                addAttributes.P.class.setOnly=blank
        }
}
```

Сравнивая первый пример со вторым, можно заметить два параметра, занимающиеся одни и тем же. Первый — "wrapNonWrappedLines", и второй — "nonWrappedTag". Разница в том, что "nonWrappedTag" может быть дополнен, а для "wrapNonWrappedLines" требуется полный тег-обертка. Если имеются уже заключенные в теги строки, вроде: | и "wrapNonWrappedLines" определено, как | , в результате появиться смесь тегов P с классами и без, вместо ожидаемого результата.

Продемонстрируем наглядно: чтобы избавиться от ненужного class="bodytext" нужно было всего лишь добавить следующую строку в настройку шаблона TS:

```
lib.parseFunc_RTE.nonTypoTagStdWrap.encapsLines.addAttributes.P.class >
```

file link

Посредством функции "file link" мы можем формировать, как подсказывает название, ссылки на файлы. Но можно сформировать не просто ссылку на файл, но и добавить значок и показать размер файла.

```
temp.example = TEXT
temp.example {

      # описание ссылки и, одновременно, название файла
      value = my_image.png

      filelink {

            # Путь к файлу
            path = fileadmin/images/

            # У файла должен быть значок
            icon = 1

            # Значок помещен в обертку
            icon.wrap = <span class="icon">|</span>

            # Значок также помещается в ссылку на файл
            icon_link = 1

            # Вместо символов типа файла, файл будет
            # отображаться как значок, если его тип png или gif
            icon_image_ext_list = png,gif

            # Кроме того, будет показан размер
            size = 1

            # Обертка для размера файла (с учетом пробелов)
            size.noTrimWrap = | (| Bytes) |

            # Размер файла будет выводиться в байтах
            size.bytes = 1

            # Аббревиатура для различных единиц размерности файлов
            size.bytes.labels =  | K| M| G

            # Обертка для всего элемента
            stdWrap.wrap = <div class="filelink">|</div>
      }
}
```

parseFunc

Эта функция анализирует основную часть содержимого, например, содержимое, введенное в Rich-Text-Editor. Функция отвечает за то, чтобы содержимое выводилось не совсем (а может даже совсем не так) так, как было введено в RTE. Некоторые правила анализа по умолчанию применялись в "css_styled_content", и некоторые из них были объяснены в главе, посвященной функции encapsLines. Если нужно изменить в TYPO3 обертку для чего-либо повторяющегося время от времени, то это можно сделать с помощью функции parseFunc. Кроме того, ее можно использовать для поиска и замены определенных строк.

В следующем примере любое вхождение "COMP" заменяется на "My company name".

```
page.stdWrap.parseFunc.short {
 COMP = My company name
 }
```

Различные возможности изменения поведения по умолчанию запросто находятся при помощи **проводника по объектам TypoScript**. Возможности настройки parseFunc можно найти здесь: parseFunc .

tags

Функция "tags" используется в комбинации с parseFunc для получения настроенных тегов. Например, в расширении "css_styled_content" определяется тег <LINK> для создания простых ссылок.

```
tags {
        # Здесь определяется название нового тега
        link = TEXT

        # далее тег анализируется/обрабатывается
        link {
                current = 1
                typolink {

                        parameter.data = parameters:allParams

                        extTarget = {$styles.content.links.extTarget}

                        target = {$styles.content.links.target}

                }

                parseFunc.constants=1
        }
}
```

Эта функция особенно полезна, если определенные типы элементов очень часто используются редакторами, и мы, тем самым сможем упростить им жизнь. Мы можем сделать так, чтобы редакторам не нужно было каждый раз форматировать это вручную, нужно лишь ввести тег, а форматирование будет сделано автоматически.

HTMLparser

HTML-анализатор определяет, как обрабатывается содержимое. Обычно используется как подфункция для parseFunc. Например, можно определить, что все ссылки будут устанавливаться с абсолютными значениями (например, для новостной рассылки):

```
page.stdWrap.HTMLparser = 1
page.stdWrap.HTMLparser {
      keepNonMatchedTags=1

      # Здесь мы определяем домен, помещаемый в начале относительного пути
      tags.a.fixAttrib.href.prefixRelPathWith=http://www.example.com/

      # Все ссылки без цели, получают target = _blank
      tags.a.fixAttrib.target.default=_blank
}
```

Функция HTMLparser очень мощная, так как перед выводом можно изменить все содержимое. Можно определить свои теги — например внутренние ссылки будут храниться так: <link http://www.typo3.org/>Linktext</link>, здесь использовать свой тег. Такие теги могут быть определены для всех полей, в том числе заголовков, для которых был определен анализатор.

Следующий пример разрешает тег <u> в заголовках. Изменяется определение по умолчанию из "css_styled_content". Функция htmlSpecialChars отключена, поэтому <u> не затрагивается. Затем используется функция parseFunc, определяющая, что нужно оставить только тег "u" и никаких других. При этом будут удалены все теги, кроме <u>.

```
# В заголовках должен быть позволен тег <u>
# Кроме этого, все элементы анализируются обычным образом
lib.stdheader.10.setCurrent.htmlSpecialChars = 0
lib.stdheader.10.setCurrent.parseFunc {
  allowTags = u
  denyTags = *
  constants=1
```

```
    nonTypoTagStdWrap.HTMLparser = 1
    nonTypoTagStdWrap.HTMLparser {
      keepNonMatchedTags=1
      htmlSpecialChars = 2
      allowTags = u
      removeTags = *
    }
}
```

Этот пример снова показывает, как важна функция stdWrap. Функция setCurrent имеет тип string/stdWrap, что позволяет использовать функцию parseFunc.

Правильное использование stdWrap

Функция stdWrap включает множество функций и параметров. Некоторые интуитивно понятны, использование некоторых сопряжено с трудностями. Здесь мы поговорим об основных принципах и выделим несколько специальных функций/методов.

Свойство stdWrap можно использовать только лишь при явном его определении. Если имеется свойство типа "wrap" то здесь нет свойств stdWrap. По умолчанию либо присутствует свойство типа stdWrap, либо свойство представлено как, например, "string/stdWrap".

```
10 = IMAGE
10.stdWrap.typolink...
```

Объект имеет свойство stdWrap типа stdWrap.

```
10 = HTML
10.value = Hello World
10.value.typolink …
```

Объект HTML отличается тем, что имеет свойство типа string/stdWrap. Мы можем добавить строку и, в дополнение, использовать свойства stdWrap.

Учет порядка следования

Должно быть подчеркнуто важное ограничение:

Функции выполняются в порядке, определенном в справочнике.

Если не уделять внимания этому факту, результат может отличаться от ожидаемого.

```
10 = TEXT
10.value = Hello World
10.case = upper
10.field = header
# подразумевается заголовок, содержащий “typo3” (символы в нижнем регистре)
10.stdWrap.wrap = <strong>|</strong>

# результат следующий:
<STRONG>TYPO3</STRONG>
```

В примере происходит следующее: первое значение объекта TEXT установлено в "Hello world". Мы знаем, что настройка TypoScripte хранится в массиве. Этот массив упорядочен не так, как TypoScript. Порядок в массиве ограничен порядком определения в stdWrap. Кинув взгляд в TSRef становится ясно, что сначала обрабатывается “field”, затем stdWrap (и внутри него “stdWrap.wrap”), а в конце “case”.

Использование stdWrap в цикле

Из-за того, что функция stdWrap может вызываться рекурсивно, возможно поменять порядок выполнения.

Функция "prioriCalc" разрешает простые математические выражения. Если установлена в 1, содержимое вычисляется, но вычисление производится слева направо (никаких математических приоритетов, вроде вычисление "*" перед "+" и т.п.). Следующий пример выглядит так, как если бы к содержимому поля "width" добавили 20.

```
10 = TEXT
10.field = width   # Предполагается: "width" равна 100
10.wrap = |+20
10.prioriCalc = 1
```

Но не в этом случае. Будет выведен результат "100+20" . Функция "prioriCalc" выполняется перед выполнением функции wrap, таким образом, в процессе вычислений "field" получает выражение "100". Для получения нужного результата нужно удостовериться, что "field" и "wrap" выполняются перед выполнением "prioriCalc". Это достижимо посредством следующего выражения:

```
10.stdWrap.wrap = |+20
```

Функция stdWrap будет выполнена после "field", но перед "prioriCalc", поэтому "100+20" обертывается, а затем выполняется "prioriCalc", в результате чего получается значение "120".

Тип данных

Работая с TypoScript критично знать, какой тип данных мы обрабатываем. Это особенно заметно в stdWrap, обычная практика использования которой — комбинировать произвольно функции, пока случайно не будет получен нужный результат.

Только при явном объявлении функционала stdWrap, можно использовать ее функции, вроде field, data или typolink.

lang: многоязычная функциональность

stdWrap предлагает свойство "lang", с помощью которого возможно перевести простые тексты, появляющиеся при помощи TypoScript на странице.

```
10 = TEXT
10.value = Imprint
10.lang.de = Impressum
10.typolink.parameter = 10
```

Но такие тексты сложно переводить внешним редакторам. Это может стать большой проблемой, особенно с незнакомыми языками.

В этом случае лучше обработать переводы через константы. Они должны быть помещены вместе и выведены при помощи TypoScript.

```
# Константы
text.imprint = Imprint
text.de.imprint = Impressum

# Настройка
10 = TEXT
10.value = {$text.imprint}
10.lang.en = {$text.en.imprint}
10.typolink.parameter = 10
```

Таким образом, перевод не зависит от TS настройки элемента.

cObject

Параметр cObject может использоваться для замены содержимого на объект TypoScript. Это может быть COA, дополнение или текст, как в этом примере:

```
10.typolink.title.cObject = TEXT
10.typolink.title.cObject.value = Impressum
10.typolink.title.cObject.lang.en = Imprint
```

Перспектива

Руководство по прежнему поддерживается в wiki, сделанные там изменения регулярно вносятся в это руководство. Если есть желание изменить что-либо, воспользуйтесь страницей wiki http://wiki.typo3.org/Ts45min, если же найдена ошибка, то сообщение о ней отправляйте на http://forge.typo3.org/projects/typo3v4-doc_tut_ts45/issues.