Синтаксис TypoScript и его всестороннее изучение

Ключ расширения: doc_core_ts

Язык: ru

Ключевые слова: typoscript, синтаксис, для администраторов, для промежуточного изучения

Пересмотрен для TYPO3 4.2, Ноябрь 2008

Содержание

Введение

Фактически, этот документ должен быть намного короче – весь *синтаксис* TypoScript можно описать в нескольких строках. Однако у TypoScript есть история, и путаница, вносимая людьми относительно него, ввиду чего эта вступительная глава должна внести ясность и уверенность, что Вы верно воспринимаете TypoScript.

Я рекомендую по крайней мере ознакомиться со следующей главой "Что такое TypoScript?», после чего переходить к главе «Синтаксис» начать читать ее. Если Вы начнете сомневаться относительно понятия TypoScript, вернитесь назад и прочитайте введение повнимательней!

Если Вы уже сомневаетесь, то ничего не пропускайте. Семь раз отмерь, один раз отрежь .

Что такое TypoScript?

Так как обычно люди сомневаются, что же такое TypoScript (TS), где он может использоваться, ввиду чего имеют склонность думать о нем, как о чем-то сложном, этот документ написан для прояснения всех этих вопросов.

Сначала начнем с нескольких прописных истин:

- TypoScript это *синтаксис* для представления информации в иерархической структуре, используя простое ASCII текстовое содержимое.

Что значит:

- Сам по себе TypoScript ничего не "делает" – он просто содержит информацию.
- TypoScript преобразуется в функции *лишь*, при поступлении в программу, разработанную с целью *выполнять действия* согласно информационной структуре, описанной TypoScript.

Таким образом? строго говоря TypoScript сам по себе не содержит функций, помимо использования в определенном контексте. Так как контекст – это почти всегда настройка чего-либо, TypoScript зачастую воспринимается как параметры (или аргументы функции) переданные функции, действующей соответственно им (напр. "background_color = red"). И напротив, Вы, вероятно, никогда не увидите, чтобы TypoScript использовался для хранениянформации, подобной базе данных адресов – для этого легче воспользоваться XML или SQL.

PHP массивы

В свете применения TypoScript может восприниматься как нестрогий способ ввода информации в *многомерный* массив. Фактически, при интерпретации TypoScript, производиться *преобразование в массив PHP*! Как бы Вы определяли статическую информацию в PHP массивах? Вы сделали бы это в файлах настройки, но точно не стали бы делать базу данных, подобной базе данных адресов!

Подведем итог:

- При *интерпретации* TypoScript, информация преобразуется в *массив PHP*, которым и пользуются приложения TYPO3
- Таким образом, *та же* информация, фактически может определятся как через TypoScript *или непосредственно в* PHP; естественно синтаксис в обоих случаях будет различным.
- TypoScript предлагает удобные возможности, поэтому мы и не определяем информацию непосредственно с синтаксисом и через массивы PHP. Среди особенностей следует отметить меньшую чувствительность к ошибкам в синтаксисе, определение значений с использованием меньшего числа символов и возможность метафорического описания обектов, их свойств и пр.

Детально смотрите в конце следующей страницы.

TypoScript синтаксис, пути к объектам, объекты и свойства

Посмотрите о чем этот документ — *синтаксис* TypoScript; правила, которых Вы должны придерживаться для хранения информации в этой структуре. Разумеется, я не буду объяснять весь синтаксис, приведу лишь примеры для передачи основной идеи.

Помните о *хранении информации*, и в этом контексте назначение TypoScript – *присвоение значений переменным*: "переменные" называют "путь объекта", потому что TypoScript хорошо вписывается в метафору "объектов" и "свойств". В этом заключены огромные преимущества, но в то же время, в TypoScript позволяет легко и просто присваивать значения; используйте для этого знак "=":

```
asdf = qwerty
```

Теперь путь объекта "asdf" содержит значение "qwerty".

Другой пример:

```
asdf.zxcvbnm = uiop
asdf.backgroundColor = blue
```

Теперь путь объекта "asdf.zxcvbnm" содержит значение "uiop", а "asdf.backgroundColor" – значение "blue". В соответствии с *синтаксисом* TypoScript запись можно упростить:

```
asdf {
  zxcvbnm = uiop
  backgroundColor = blue
}
```

Здесь мы разделили полный *путь объекта* "asdf.zxcvbnm" на компоненты, разделенные точкой ".", "asdf" и "zxcvbnm", а для их связи, использовали оператор фигурные скобки, { и }. Для описания отношений компонентов *пути объекта*, обычно "asdf " называют *объектом*, а " zxcvbnm " его *свойством*.

Так, хотя термины *объекты* и *свойства* обычно воспринимаются в неком контексте (семантика), можно их использовать просто для описания различных частей пути объекта не вкладывая определенный контекст и значение. Посмотрите на это:

```
asdf {
  zxcvbnm = uiop
  backgroundColor = blue
  backgroundColor.transparency = 95%
}
```

Можно сказать, что "zxcvbnm" и "backgroundColor" – это *свойства* (объекта) "asdf". Далее, "transparency" – это свойство (объекта/свойства) "backgroundColor" (или объекта "asdf.backgroundColor").

Примечание относительно восприятия (семантики):

Можно подумать, что в "backgroundColor = blue" больше смысла, чем в "zxcvbnm = uiop", но это не так! Единственная причина, по которой в "backgroundColor = blue" мы видим больше смысла, – это *английский язык* (если Вы его понимаете) в котором слова "background color" и "blue" автоматически подразумевают некое значение. Но для машины (компьютера), слово "backgroundColor" имеет не больше смысла, чем "zxcvbnm", конечно, если она не была запрограмирована на понимание его, как значения цвет фона чего-либо. Фактически, "uiop" может быть псевдонимом значений синего цвета, а "zxcvbnm" – свойство цвета фона чего либо, например страницы HTML.

Это просто служит указанием на одну вещь: хотя TypoScript, как и большинство языков программирования, используют названия функций, методов, свойств ключевые слова понятные для людей, но все же в конечном счете эти выражения определяются в справочнике, DTD или XML-схеме.

Примечание о внутренней структуре при интерпретации в массив PHP:

В начале предыдущей главы, TypoScript позиционировался как легкий способ ввода информации в многомерный массив PHP. Давайте возьмем в качестве примера следующий TypoScript

```
asdf {
  zxcvbnm = uiop
  backgroundColor = blue
  backgroundColor.transparency = 95%
}
```

При интерпретации, эта информация сохраняется в массиве PHP, который можно определить так:

```
$TS['asdf.']['zxcvbnm'] = 'uiop';
$TS['asdf.']['backgroundColor'] = 'blue';
$TS['asdf.']['backgroundColor.']['transparency'] = '95%';
```

Или альтернативно можно определить инфромацию в массиве PHP так:

```
$TS = array(
    'asdf.' => array(
        'zxcvbnm' => 'uiop',
        'backgroundColor' => 'blue',
        'backgroundColor.' => array (
            'transparency' => '95%'
        )
    )
)
```

Лично я убежден, что TypoScript обладает лучшим синтаксисом для работы! Пользователь, не имея технического склада ума сможет легко написать скрипт с минимальным количеством символов (хотя *TS нацелен на технарей*!) не боясь ошибок интерпретации PHP, приводящим к прекращению формирования страницы (или чего-либо).

Синтаксис

Введение

Помните, TypoScript поход на (большой) многоразмерный PHP массив; значения расположены в иерархии – дереве. "Ветви" обозначены точками (".") – синтаксис, похожий, например на JavaScript, в основе лежит идея определения объектов/свойств. Дополнительная информация об этой метафоре содержится во вступительной главе этого документа. Здесь мы будем иметь дело лишь с правилами синтаксиса (написания).

Пример:

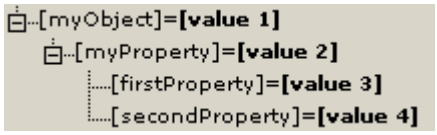
```
myObject = [value 1]
myObject.myProperty = [value 2]
myObject.myProperty.firstProperty = [value 3]
myObject.myProperty.secondProperty = [value 4]
```

Иными словами, об объекте "myObject", можно сказать: "*объект со значением [value 1] и свойством, 'myProperty' значение которого [value 2]. Также 'myPropery' само имеет два свойства, 'firstProperty' и 'secondProperty' со значениями, соответственно ([value 3] и[value 4]).*"

Можно даже вызвать 'myProperty' для TypoScript объекта этого примера! Это уже объяснялось во вступлении.

Помните, TypoScript содержит информацию, а не "выполняет(делает)" что-то, как настоящий язык сценариев; поэтому в нем нет сообщений об ошибках при определении несуществующих объектов или свойств. Таким образом TypoScript можно оценивать лишь при его использовании в определенном контексте (например "Шаблоны TypoScript" или "Tsconfig страницы"). Однако мы всегда можем оценить правильное использование TypoScript со стороны синтаксиса (что значит, насколько правильно определена информация с точки зрения "грамматики").

В любом случае, этот TypoScript можно представить как дерево с узлами (Как в проводнике объектов TypoScript):



Синтаксис TypoScript

TypoScript обрабатывается очень просто; строка за строкой. Каждая строка обычно состоит из трех частей:

```
[Путь объекта] [Оператор] [Значение]
```

Пример:

```
myObject.myProperty = [value 2]
```

- Путь объекта соответствует имени переменной в языке программирования (Здесь: myObject.myProperty). Путь объекта – первый блок символов строки без пробелов до символов "=<>{(" (включая пробел). **Для пути объектов используйте только A-Z, a-z, 0-9, "-", "_" и точки (.).**
- Оператор – один из символов: "=<>{(" (Здесь: "="). Значение каждого описано ниже.
  There is one special operator «:=» that changes the value depending on a predefined function.
- Значение – любые символы после оператора до конца строки, обрезаемые пробелом (Здесь: [value 2]). Помните –, значения не нужно помещать в кавычки! Значение начинается после оператора и заканчивается концом строки.

Пример

Здесь myObject определен как объект-содержимое HTML с установленным свойством "value" (в контексте Шаблонов TypoScript):

```
myObject = HTML
myObject.value = <BLINK> HTML - code </BLINK>
```

Комментарии: когда строка начинаестя с "/" или "#"

...это обозначает комментарий, и строка полностью игнорируется.

Пример:

```
// Это комментарий
/ Это тоже комментарий (можно применять лишь ОДИН слеш)
myObject = HTML
myObject.value = <BLINK> HTML - code </BLINK>
# Эта строка — тоже комментарий.
```

Блок комментария: когда строка начинается с "/*" или "*/"

... это обозначает, соответственно, начало или конец раздела комментария. Все содержимое раздела комментария игнорируется.

Правила:

- /* и */ ДОЛЖНЫ стоять первыми в строке для их правильного опознания.
- Комментарии не определяются внутри многострочного значения, заключенного в круглые скобки.

Пример:

```
/* Это комментарий
 .. и эта строка тоже в пределах блока комментария...
  а вот конец блока:
*/  ... это тоже не обрабатывается - вся строка находится в пределах блока комментария
myObject = HTML
myObject.value (
  Это многострочное значение, в котором
  /*
    комментарии игнорируются, и эта строка будет интерпретирована!
  */
)
```

Присвоение значения: оператор "="

Присвоение значения пути объекта.

Правила:

- Все после знака "=" *до конца строки* является значением. Другими словами: ничего не нужно заключать в кавычки! Удостоверьтесь, что значение не обрезается, т.е. с обоих сторон имеются пробелы.

Изменение значения: оператор ":="

Присвоение значения пути объекта, вызывая определенную функцию, изменяющую различным образом существующее значение пути объекта.

Полезно для списков ID страниц и т.п., когда значение должно быть расширено без полного переопределения.

Правила:

- Часть после оператора ":=" *до конца строки* разбивается на две части: функция и значение. Функция определяется правой, близкой к оператору частью (обрезается) и заключенным в скобки значением (не обрезается).
- Существует четыре предопределенные функции:
  - **prependString:** добавляется строка к *началу* существующего значения.
  - **appendString:** добавляется строка к *концу* существующего значения.
  - **removeString:** удаляет строку из существующих значений.
  - **replaceString:** заменяет старые значения на новые. Используется разделитель «|».
  - **addToList:** добавляется список значений через запятую к концу значения строки. Проверка на дублирование значений не проводится и список никак не сортируется.
  - **removeFromList:** удаляется список значений через запятую.
- Внутри class.t3lib_tsparser.php имеется обработчик, который можно использовать для определения подобных функций.

Пример:

```
myObject = TEXT
myObject.value = 1,2,3
myObject.value := addToList(4,5)
myObject.value := removeFromList(2,1)
```

...результат аналогичен следующему:

```
myObject = TEXT
myObject.value = 3,4,5
```

Ограничения: знаки { }

(открывающая/закрывающая фигурные скобки) значит, что Вы хотите просто за раз назначить множество свойств объекта. Это называется *ограничение* или вложение свойств.

Пример:

```
myObject = HTML
myObject.value = <BLINK> HTML - code </BLINK>
```

...то же, что и:

```
myObject = HTML
myObject {
  value = <BLINK> HTML - code </BLINK>
}
```

Правила:

- Все, что идет после "{" во второй строке игнорируется.
- Знак "}" *должен* начинать строку без пробелов до него, в случае окончания назначия свойств. Любые символя после знака "}" игнорируются.
- **ПРИМЕЧАНИЕ:** в фигурных скобках нельзя использовать условия (за исключением [GLOBAL] условие, которое будет обнаружено и уровень фигурных скобок будет сборошен на ноль)
- **ПРИМЕЧАНИЕ:** несколько конечных фигурных скобок игнорируются, но в синтаксическом анализаторе TypoScript появится предупреждение.

Многострочное значение: знаки ( )

(открывающая/закрывающая скобка) значит, что Вы можеде присвоить *многострочное значение*. Так Вы можете определить многострочные значения, включая разрывы строк.

Пример:

```
myObject = HTML
myObject.value (
  <BLINK>
    HTML - code
  </BLINK>
)
```

Правила:

- **ПРИМЕЧАНИЕ:** очень важна закрывающая скобка, если она не будет найдена, анализатор не вернется к анализу TypoScript. Это относится и к [GLOBAL] условиям, которые Вам не помогут! Не забывайте!

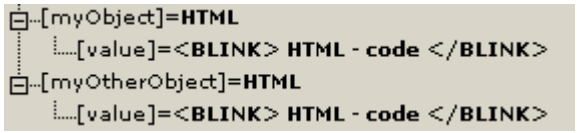
Копирование объектов: знак "<"

Используется для копирования одного пути объекта в другой. Копируется целый объект со значениями и свойствами, замещая любые аналогичные старые объекты и свойства.

Пример:

```
myObject = HTML
myObject.value = <BLINK> HTML - code </BLINK>

myOtherObject < myObject
```

В этом случае получим два независимых дублирующих набора объектов/свойств. Они *не* зависят друг от друга, а являются лишь копиями:



Другой пример (копия с использованием фигурных скобок):

```
pageObj {
  10 = HTML
  10.value = <BLINK> HTML - code </BLINK>
  20 < pageObj.10
}
```

Здесь также копируется объект. Объект определен от корня. Вы можете обозначить общий корень. При этом не нужно писать название корневого объекта, нужно всего лишь поставить точку для обозначения принадлежности тому же уровню.

Это выражение эквивалентно предыдущему:

```
pageObj {
  10 = HTML
  10.value = <BLINK> HTML - code </BLINK>
  20 <.10
}
```

Примечание о ссылках на объекты (в Шаблонах TypoScript):

Когда TypoScript используется в контексте Шаблонов TypoScript, можно обнаружит, что на объекты содержимого иногда можно ссылаться вместо копирования. Ссылки означают, что множество объектов дерева могут использовать один и тот же объект, без создания фактической его копии, просто указывая на его полный путь.

Пример на основе предыдущего кода:

```
0: myObject = HTML
1: myObject.value = <BLINK> HTML - code </BLINK>
2:
3: pageObj {
4:   10 = < myObject
5: }
```

Присмотритесь к строке 4: похоже на свойство TypoScript, ***но это не так!!!*** Эта возможность появляется на уровне контекста, то есть это будет работать только потому, что движок Шаблонов TypoScript запрограммирован на распознавание названий объектов содержимого, начинающихся с "<", как ссылок на пути объектов полс знака <.

Фактически, если посмотреть на разобранный TypoScript в дереве объектов, можно увидеть, что "< myObject" – ничто иное, как путь к объекту "pageObj.10":

Это примечание нужно во избежание неправильного представления, которое Вы могли бы получить, когда начинали работать с TypoScript для построения шаблонов.

Сброс установок объекта: знак ">":

Используется для сброса объекта и все его свойств.

Пример:

```
myObject = HTML
myObject.value = <BLINK> HTML - code </BLINK>

myObject >
```

В последней строке, объект "myObject" полностью стирается (удаляется)

Условия: строка, начинающаяся с «[«

Условия прерывают интерпретацию TypoScript в зависимости от значения содрежимого строки условия. Если условия соответствуют, интерпретация продолжается, иначе TypoScript игнорируется, пока не встретиться другое условие. В следующем разделе документа детально разбираются условия.

Пример:

```
[browser=netscape]
page.10.value = Netscape

[else]
page.10.value = Not a netscape browser!

[end]
```

Правила:

- Условия могут прервать интерпретацию, *только* вне любых ограничений (уровня фигурных скобок).

Условия

Возможно использование, так называемых *условий* в TypoScript. Условия – простые управляющие структуры, проверяющие соответствие, некоторым критериям (внешним), и таким образом определяющим, нужно ли разбирать TypoScript до следующего найденного условия.

Примеры условий:

- Это "Netscape" браузер?
- Установлена ли группа пользователей для текущей сессии?
- Сегодня понедельние?
- Установлен ли параметр GET "&language=uk"?
- Сегодня день рождения моей матери?
- Я сегодня счастлив?

Из этих примеров, первый – наиболее реалистичный в контексте ШаблоновTypoScript. Но теоретически, условие может оценивать соответствие любому обстоятельству и возвратить либо ложь, либо истину, отражающуюся на интерпретации дальнейшего кода TypoScript.

Где можно использовать условия

Определение условий – часть синтаксиса TypoScript, но *проверка правильности* содержимого условия всегда зависит от контекста использования TypoScript. Поэтому, в плане выделения синтаксиса (без учета контекста), условия просто выделяются и ничего более. В контексте Шаблонов TypoScript существует целый раздел Tsref, определяющий синтаксис содержимого условий для Шаблонов TypoScript. Для "Tsconfig страницы" и "Tsconfig пользователя" условия вообще не реализованы.

Синтаксис условий

Условия всегда расположены на отдельной строке, котороая определяется по "[" (квадратной скобке) – первому символу этой строки:

```
(какой-то TypoScript)

[ condition 1 ][ condition 2]

(Некий TypoScript, интерпретируемый при выполнении условия 1 или условия 2)

[GLOBAL]

(Какой-то TypoScript)
```

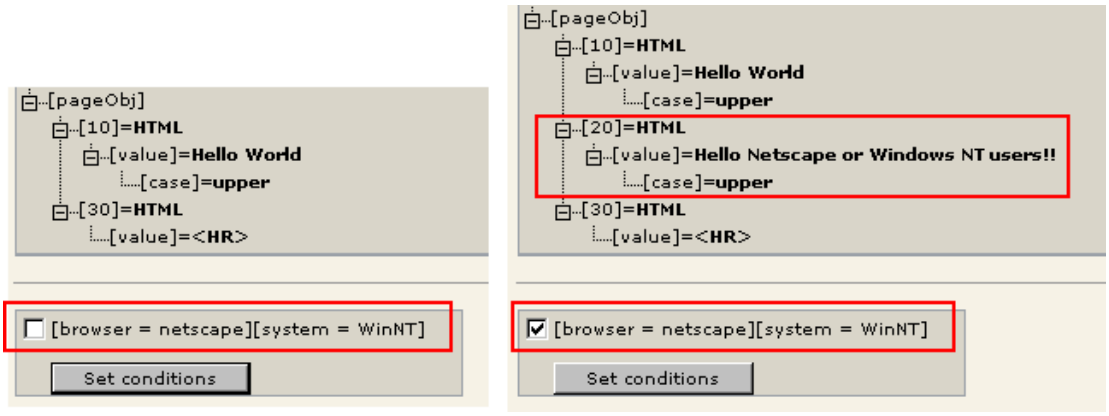
Так, в этом примере, строка "[GLOBAL]" – это условие (встроенное, всегда истинно), так же, как и строка "[condition 1][condition 2]". Если "[condition 1][condition 2]" истинно, то идущий после TypoScript будет прочитан до [GLOBAL] (или [END]), условия будут сброшены и чтение последующего TypoScript возобновится.

Примечание: в строке "[condition 1][condition 2]" установлены *два условия*, но с точки зрения синтаксических анализаторов TypoScript, условием является *целая строка* – в контексте Шаблонов, строка дробится на меньшие единицы ("[condition 1]" и "[condition 2]"), каждое из которых оценивается и посредством операнда ИЛИ сливаются вместе, перед возвращением результирующей истины или лжи. (За это отвечает класс t3lib_matchCondition).

Вот пример некоего TypoScript (в контексте Шаблонов TypoScript), выводящего другой текст, если используется веб-браузер Netscape (вместо, например Internet Explorer) или операционная система – Windows NT:

```
pageObj.10 = HTML
pageObj.10.value = Hello World
pageObj.10.value.case = upper

[browser = netscape][system = WinNT]
pageObj.20 = HTML
pageObj.20 {
  value = Hello Netscape or Windows NT users!!
  value.case = upper
}

[GLOBAL]
pageObj.30 = HTML
pageObj.30.value = <HR>
```

Можно, используя Проводник объектов, увидеть разницу в обработанном дереве объектов, в зависимоти от соответствию условий (что можно симулировать в этом модуле, как Вы можете заметить):



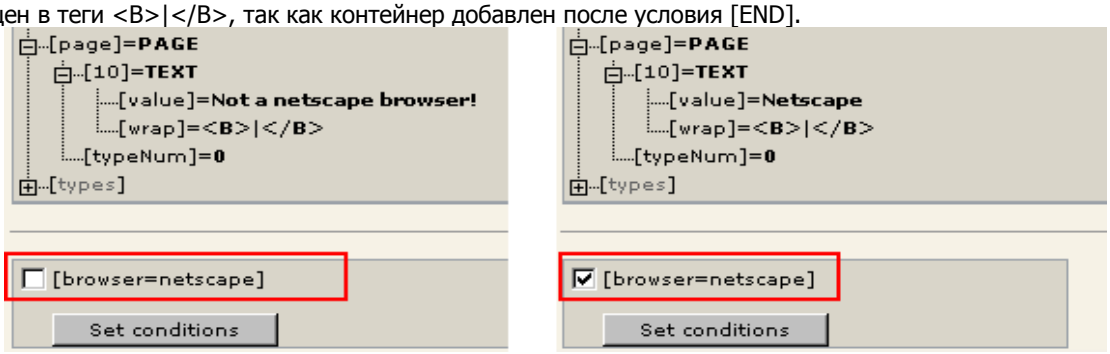
Специальные [ELSE], [END] и [GLOBAL] условия

Существует специальное условие [ELSE], возвращающее истину, если предыдущее условие возвратило ложь. В завершение условия [ELSE] можно использовать либо [END], либо [GLOBAL] условия. Все три условия могут быть записаны и в нижнем регистре.

Вот пример использования [ELSE]-условия (также в контексте Шаблонов TypoScript):

```
page.typeNum=0
page = PAGE
page.10 = TEXT

[browser=netscape]
page.10.value = Netscape

[else]
page.10.value = Not a netscape browser!

[end]

page.10.wrap = <B>|</B>
```

При этом, один текст будет выведен в браузере Netscape, а другой в остальных браузерах. В любом случае, текст будет помещен в теги <B>|</B>, так как контейнер добавлен после условия [END].



Фактически, "включение" условия в Проводнике объектов TypoScript, симулирует выполнение любого из условий, включенного в Шаблон Template. Соответствие условий (в этом примере) проверяется лишь по браузеру Netscape, в котором открывается страница.

Другой пример может использоваться для выполнения чего-либо в случае невыполнения нескольких условий. Несмотря на отсутствие знаков отрицания, это можно сделать:

```
[browser=netscape][usergroup=3]
  # Здесь ничего не вводите!
[else]
  page.10.value = Этот текст отобразиться, только при Невыполнении условий выше!
[end]
```

Где в TypoScript вставлять условия?

Условия можно вставить только *вне ограничений* (фигурных скобок)!

Это правильно:

```
someObject {
  1property = 234
}
[browser=netscape]
someObject {
  2property = 567
}
```

А это **нет:**

```
someObject {
  1property = 234
  [browser=netscape]
  2property = 567
}
```

```
someObject {
  1property = 234
  [browser=netscape] - ERROR: Line 2: Object Name String, "[browser" contains invalid character "[".
  2property = 567
}
```

При интерпретации с подстветкой синтаксиса, Вы увидите следующую ошибку: это значит, что строка воспринимается как обычное определение "[путь объекта] [оператор] [значение]", а не как условие.

Условие [GLOBAL]

Однако для специального условия [GLOBAL] (отменяющее любую предыдущую условную область), правило немного отличается, так как оно будет обнаружено в *любой строке* за исключением многострочной.

```
someObject {
  1property = 234
  [GLOBAL]
  2property = 567
}
```

Все равно обнаружите некую ошибку при подсветке синтаксиса:

```
someObject {
  1property = 234
[GLOBAL] - ERROR: Line 2: On return to [GLOBAL] scope, the script was short of 1 end brace(s)
  2property = 567
} - ERROR: Line 4: An end brace is in excess.
```

Причина этого в том, что условие [GLOBAL] прерывает ограничение, начинающееся в первой строке, приводя к первой ошибке («... short of 1 end brace(s)»). Вторая ошибка появляется из-за ненужной закрывающей фигурной скобки, так как "уровень ограничения" был отменен условием [GLOBAL].

Итак, подытожим; специальное условие [global] (или [GLOBAL]) всегда прерывает интерпретацию TypoScript внутри фигурных скобок, переходя в общую область (если не присутствует в многострочном значении). Это работает в любой реализации TypoScript, независимо от возможности применения других типов условий. Поэтому, можно использовать [GLOBAL] (записанное в одну строку), чтобы удостовериться, что выполнение TypoScript продолжено на общем уровне. Обычно это делается при использовании комбинации TypoScript кода из различных записей.

Резюме

- Условия выявляются по символу "[", стоящему в начале строки (пробелы игнорируются).
- Условия оцениваются относительно контекста, в котором используется TypoScript; они широко распространены в Шаблонах TypoScript, но совсем не используются в “Tsconfig страницы” или “Tsconfig пользователя”.
- Существуют специальные условия [ELSE], [END] и [GLOBAL].
- Условия можно употреблять только вне ограничений (фигурных скобок). Однако условие [GLOBAL], всегда прерывает ограничение, если употребляется в одиночной строке.

Включения

Также можно включить инструкции в код TypoScript. Доступность зависит от контекста, но работает с шаблонами TypoScript, Tsconfig Страницы и Tsconfig Пользователя.

Включение инструкций выглядит так:

```
<INCLUDE_TYPOSCRIPT: source="FILE: fileadmin/html/mainmenu_typoscript.txt">
```

- Должно находиться на одной строке шаблона TypoScript, иначе не будет распознано.
- Обрабатыватеся ПЕРЕД интерпретацией любого TypoScript (вопреки условиям) и, вследствии этого, не нужно заботится о месте размещения в коде TypoScript.

Параметр “source” указывает на источник включаемого содержимого. Строка перед первым ":"(в примере — слово “FILE”) определяет тип источника содержимого. Этот параметр:

FILE	Ссылка на файл, относительно PATH_site. Должен быть меньше 100 KB. Не пожет содержать «..» (две точки, путь обратно). Если в относительном пути содержится приставка, напр. “EXT:myext/directory/file.txt”, тогда поиск включаемого файла будет осуществляться в директории расширения “myext”, поддиректории “directory/file.txt”.

Детальный разбор

Дополнительно о синтаксисе, семантике. TypoScript в сравнении с XML и XSLT.

Если Вы думаете, что уже отлично понимаете TypoScript (уже может быть и так...), этот раздел не для Вас. Я рискую снова смутить Вас. Но, в любом случае, здесь собрана более теоретическая информация о TypoScript, включая и его связь с XML и XSLT:

XML и TypoScriptсхожи в синтаксисе:

Кусок кода TypoScript похож на документ XML – та же информация о структуре, ничего более. Но относительно хранения информации, как в TypoScript, так и в XML, нужно придерживаться *синтаксиса* – правил, относительно того, *как* значения вставляются в структуру. Как грамматика для разговорного языка определяет в какой последовательности и как могут комбинироваться слова.

Для XML существуют такие правила: "все теги должны закрываться, напр. ... или
", правила вложения, использовать нижний регистр для элементов и атрибутов имен и т.п. Если XML документ следует этим правилам, он называется "правильно сформированым". Для TypoScript существуют подобные правила: "оператор = задает слудущее за ним, до конца строки, как значение пути объекта" или "строка, начинающаяся с # или / – комментарий и ее содержимое игнорируется".

XSLT и "Шаблоны TypoScript" схожи в семантике (значения, функции):

Это синтаксис, соответствующий XML:

```
<asdf>qwerty</asdf>
```

А это синтаксис соответствует TypoScript:

```
asdf = qwerty
```

А это синтаксис английского языка:

```
footballs sing red
```

Но ни один из этих примеров не имеет значения без некоторого комментания, говорящего, как элементы, значения или слова могут быть объединены, чтобы приобрести *значение* – нужен *контекст.* Все это называется *семантика*. Для разговорных языков мы понимаем ее интуитивно, потому что мы знаем, что футболисты не поют и мы не можем "петь красный" – это не имеет смысла, хотя предложение правильно построено. В XML документе есть его DTD (определение типа документа) или определенная существующим на том же уровне элементом "<asdf>" схема, а для TypoScript – это *справочник* для контекста, в котором используется синтаксис TypoScript для определения информационной иерархии, например "TSref" для Шаблонов TypoScript или документ "TSconfig" для "Tsconfig страницы" или "Tsconfig пользователя".

Так, смысл в XML документе появляется только если Вы знаете связи в хранящейся в документе информации и это нужно для преобразование, через XSLT шаблон, одного XML в другой. Фактически, шаблон XSLT, что-то вроде "вавилонской рыбки" для XML – переводить один "язык" XML в другой "язык" XML.

Так и *синтаксис* TypoScript используется для построения "Шаблонов TypoScript" (*содержание семантики - значение*); информация получает смысл, если соблюдаются правила, описанные в документе "TSref".

Кстати, сравнение "Шаблонов TypoScript" и "XSLT" интенсионально, так как оба могут быть описаны, как *декларативные языки программирования* – программирование посредством *значений, инструктирующих настоящую процедурную программу* (напр. Движок внешнего представления TypoScript, который написан на PHP) как выводить данные. Подробности ищите в этой статье на typo3.org.

Подробнее о синтаксисе и семантике можно прочитать по этой ссылке, найденной в сети.

Где используется TypoScript?

На этот вопрос нельзя ответить полностью, так как этот документ описывает лишь синтаксис TypoScript, а не каждый из контекстов, в котором этот синтаксис применяется для настройки; теоретически, кто угодно может использовать обработчик TypoScript в TYPO3 для настройки своих расширений используя свою "семантику" (описано в другом разделе этого документа).

Но для начала нужно понять три основные приложения, в которых используетя синтаксис TypoScript в ядре TYPO3:

- TSconfig страниц – настройка ветвей дерева страниц.
- Tsconfig пользователей – настройка пользователей и их групп.
- Шаблоны TypoScript – описание и настройка каждого вебсайта в дереве страниц.

Tsconfig страниц

У каждой записи страницы в TYPO3 имеется поле, в которое можно ввести "TSconfig". Основная преследуемая TSconfig страницы идея – индивидуальная настройка каждой части дерева страниц. Эта возможно, так как код TypoScript введенный в поля Tsconfig, накапливается для всех страниц в корневой линии с текущей позиции страницы, начиная от корня и ниже. Параметры TypoScript в полях TSconfig внешних страниц могут переопределить те же параметры, определенные в страницах, ближе к корню.

Например, может быть два разных вебсайта, в разных ветвях дерева страниц. Один вебсайт поддерживает содержимое лишь в "обычной" колонке, а другой – как в "обычной", так и "боковой". Так как модуль страниц по умолчанию показывает все четыре возможные колонки, нужно проинструктировать модуль странц показать лишь обычную колонку и, соответсвенно, обычную + боковую. Но это возможно лишь если где-нибудь подсказать системе, что начиная с этой страницы и далее нужно использовать только "обычную" колонку, а начиная с другой страницы и после нее, использовать "обычную"+ "боковую" колонки. Так, в корневой странице двухколоночного вебсайта, нужно ввести следующую строку в поле TSconfig:

И так же для одноколоночного вебсайта нужно ввести другое значение в то же поле TSconfig его корневой страницы:

Теперь модуль "Страница" получит значение свойства "colPos_list" при отображении содержимого страниц для всех подстраниц корневой страницы, с которой была введена настройка. Соответственно, будет показаны лишь колонки, для которых была сделана настрока.

Объект и свойства, которые Вы можете здесь использовать описаны в документе "TSConfig" в дополнение к документам используемых расширений.

Tsconfig пользователей

Каждый пользователь внешнего и внутренного интерфейсов и их группы имеют поле для ввода TSconfig. Основная идея для TSconfig пользователей – возможность индивидуальной настройки как групп, так и отдельных пользователей. Это обычные настройки, но более детальные, чем те, что вы обычно устанавливаете в формах, для параметров доступа пользователей и их групп. Например, вы можете установить, сколько буферов обмена может видеть пользователь, или будет ли в формах для этого пользователя присутствовать кнопка "Сохранить и создать новый документ" – те детали, которые забили бы форму основных настроек для пользователей и их групп.

Как и в TSconfig страниц, содержимое полей TSconfig накапливается в определенном порядке; в порядке членства в группах и окончательно – настройки для конкретного пользователя. Так, настройки для пользователя перепишут теже настройки для группы, в которой он состоит.

Вот пример того, что можно сделать в TSconfig пользователя внутреннего интерфейса, эта строка добавляет кнопку "Сохранить и создать новый документ" в форму редактирования:

Объект и свойства, которые Вы можете здесь использовать описаны в документе "TSConfig" в дополнение к документам используемых расширений.

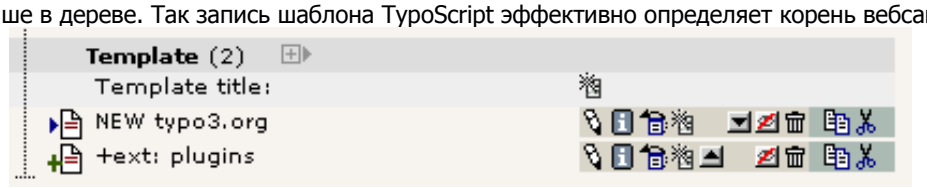
Шаблоны TypoScript

Наиболее известное и широкое употребление TypoScript получил для настройки движка внешнего отображения содержимого сайта, где TypoScript используется для определения того, как движок отобразит вебсайт. Именно здесь TypoScript пересекается с идеями традиционного построения шаблонов в веб дизайне и в представлении людей TypoScript ассоциируется с языком программирования – после чего он их еще больше пугает. (Если в TYPO3 и есть язык программирования, то это *не* TypoScript, а PHP!)

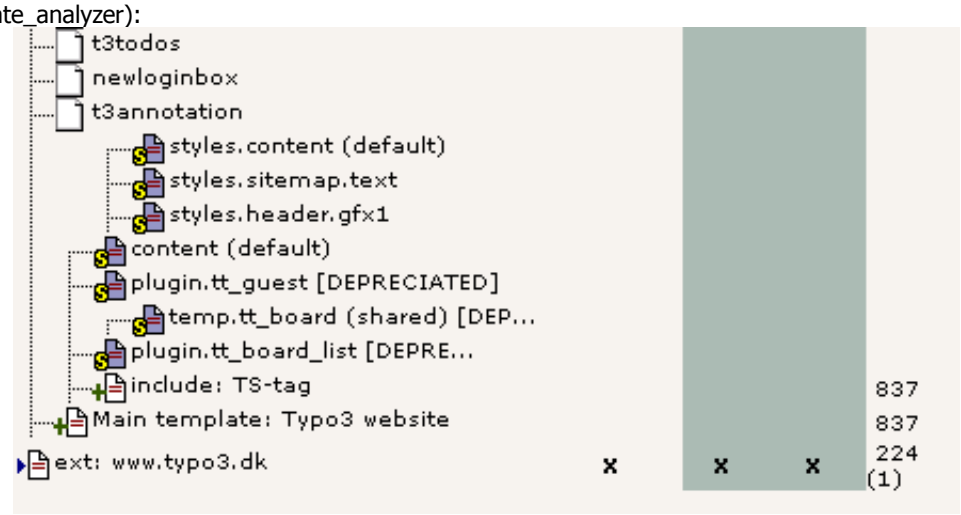
В этой вводной в TypoScript постараемся развеять ошибочные представления, для этого сделаем два замечания о том, как в TYPO3 обрабатываются шаблоны:

- **Нет строго определенных шаблонных методов:** TYPO3 *не* предлагает какой-то *один определенный* метод работы с шаблонами сайтов; наоборот, Вы *свободны* в выборе наиболее удобных методов. Вы можете:
  - **HTML шаблоны;** настройте TYPO3 для совмещения маркеров и разделов внешнего HTML шаблона. Популярно и знакомо многим людям. Ознакомьтесь с руководством "Современная разработка шаблонов, часть 1".
  - **Расш. Движок шаблонов:** настройте TYPO3 для использования шаблонов стилей XSLT для обработчика XSLT. Это можно сделать через расширение, предоставляющее такие функции, либо написать собственное расширение.
  - **Свой PHP:** настройте TYPO3 для вызова своего кода PHP, формирующего содержимое страницы любым доступным способом. Можно использовать дополнительные движки шаблонов!
  - **Объекты содержимого TS:** или формирование страницы Движком Внешнего интерфейса посредством "объектов содержимого", *доступных для программирования* через синтаксис TypoScript.
- **TypoScript Templates *determines* the method:** No matter which template method (see list above) you would like to use TYPO3 needs to be told *which one*! And *this* is what the TypoScript Template does first and foremost; it is used to configure basic and advanced behaviors of the frontend engine so that the site rendering gets done.

Шаблоны TypoScript работают почти как Tsconfig страниц; это запись базы данных, закрепляющая содержимое TypoScript за определенной страницей и следующими за ней страницами, настройка, содержащаяся в TypoScript будет применяться к страницам, до тех пор, пока не будет найдена новая запись, переписывающая содержимое страницы, расположенной выше в дереве. Так запись шаблона TypoScript эффективно определяет корень вебсайта:



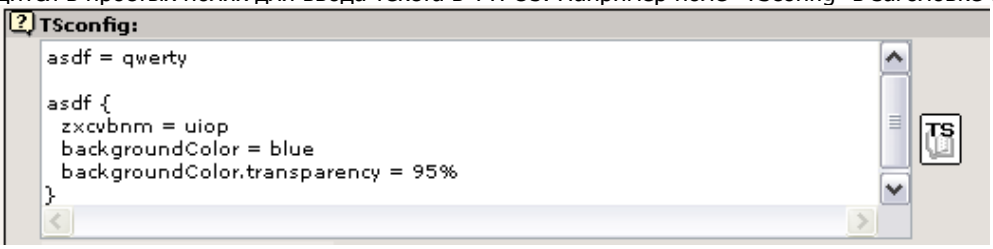
В Шаблонах TypoScript существует поле для конфигурационного кода TypoScript (поле "Setup"), но запись шаблона, подобно изображенной на рисунке выше ("NEW typo3.org") может включать ссыки на другие записи шаблонов и файлы, содержащие предопределенный общий TypoScript код, включаемый и таким образом используемый в шаблонах. Порядок подключения записей шаблонов/файлов TypoScript визуально представлен в расширении Анализатор шаблонов (tstemplate_analyzer):



Для получения дополнительных знаний о том, как работают шаблоны TypoScript, читайте в документе Шаблоны TypoScript. Объект и свойства, которые Вы можете здесь использовать описаны в справочнике по TypoScript – TSref. Практические примеры для обучения в процессе разработки, Вы найдете в TypoScript на примерах.

Ввод TypoScript

Так как TypoScript имеет строчный синтаксис, напоминающий большинство простых текстовых конфигурационных файлов, он вводится в простых полях для ввода текста в TYPO3. Например поле "TSconfig" в заголовке страницы:



Обычно TypoScript вводится непосредственно в поля формы. Без подстветки синтаксиса, без использования табуляций, без нумерации строк и т.п. Весьма скучно и просто. Конечно, эти возможности отсутствуют не по политическим соображениям, скорее невозможность их простой реализации в поле ввода обычного текста. Если кто-нибудь сможет предоставить хороший Java-редактро, мы были бы счастливы. (тем не менее, Вы можете найти поддержку подсветки синтаксиса TypoScript для многих текстовых редакторов. Пришлите запрос на почту или поищите на typo3.org).

Тем не менее есть несколько полезных возможностей:

- a) Значек "TS мастер", зачастую присутствующий справа от области ввода текста, – поможет при поиске свойств для вводимого содержимого TypoScript.
- b) Существует возможность вставить включающий тег в любое поле поле TypoScript (описывается дальше в этом документе), ссылающийся на внешний файл, в котором содержится TypoScript, а этот файл можно отредактировать любым внешним редактором.

Анализ, хранение и выполнение TypoScript

Анализ TypoScript

Это значит, что текстовое содержимое TypoScript преобразуется в структуру массива PHP по правилам синтаксиса TypoScript. Но все же, содержимое интерпретированного содержимого не оценивается.

При интерпретации, могут присутствовать синтаксические ошибки, когда введеный TypoScript текст не соответствует правилам синтаксиса TypoScript. Анализатор в этом случае очень лоялен, он лишь регистрирует ошибку, продолжая обрабатывать код TypoScript. Тем не менее, синтаксические ошибки можно увидить лишь при помощи специального анализирующего инструмента, например подсвечивающего синтаксис.

Для анализа содержимого TypoScript используется класс "t3lib_tsparser". Детали можно найти в другом разделе этого документа.

Сохранение проанализированного TypoScript

После анализа TypoScript хранится как *массив PHP* (часто преобразованный в последовательную форму и позже буферизуется в базе данных). Взяв TypoScript из примера во введении и проанализировав его, получим похожий результат:

Изначальный TypoScript:

```
asdf = qwerty
asdf {
  zxcvbnm = uiop
  backgroundColor = blue
  backgroundColor.transparency = 95%
}
```

После его разбора функцией "parse()" из класса t3lib_tsparser, внутренняя переменная $this->setup этого класса будет содержать массив PHP похожий на этот (получен через PHP функцию print_r()):

```
Array
(
    [asdf] => qwerty
    [asdf.] => Array
        (
            [zxcvbnm] => uiop
            [backgroundColor] => blue
            [backgroundColor.] => Array
                (
                    [transparency] => 95%
                )

        )

)
```

Также можно распечатать массив через API функция в TYPO3, с названием t3lib_div::view_array() или just debug(). Он будет выглядет примерно так:

asdf	qwerty
asdf.	zxcvbnm: uiop backgroundColor: blue backgroundColor.: transparency: 95%

Как видно, значение ("blue") свойства "backgroundColor" можно выбрать через этот PHP код:

```
$this->setup['asdf.']['backgroundColor']
```

Так, можно сказать, что TypoScript предлагает *интерфейс* на основе текста для получения значений многоразмерным массивом PHP из простых текстовых полей или файлов. Это очень полезно, если Вы хотите взять данные от пользователя, не давая ему непосредственного доступа к коду PHP – причина появления TypoScript.

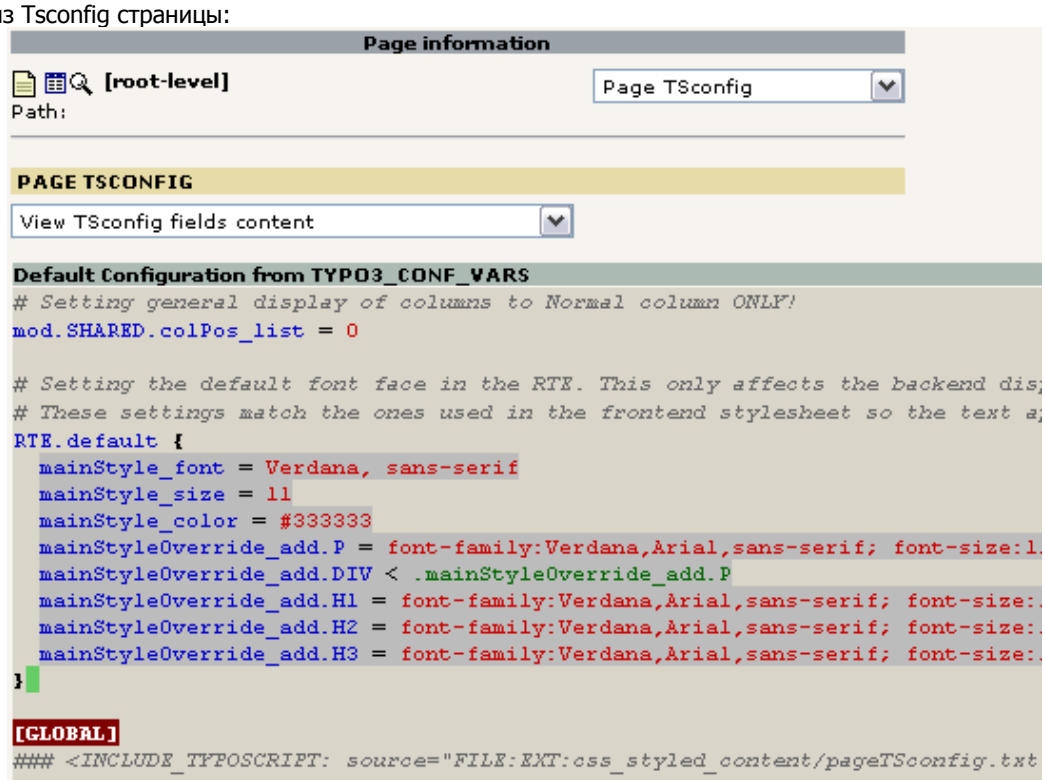
"Выполнение" TypoScript

Так как сам по себе TypoScript всего лишь содержит информацию (!), его нельзя "выполнять". Ближе всего к "исполнению" TypoScript массив PHP с разобранной структурой TypoScript, введенной в PHP функционал, который *затем* действует согласно значениям в массиве. Но мы опять возвращаемся к спорам относительно синтаксиса/семантики.

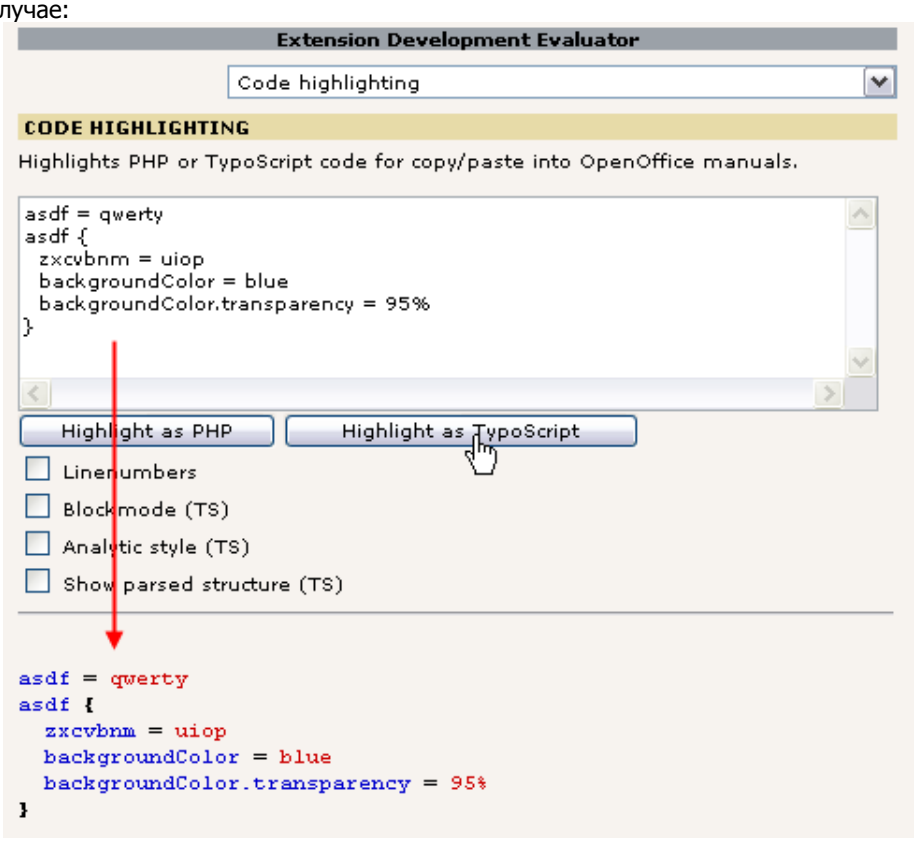
Подсветка синтаксиса и отладка

Подсветка синтаксиса кода TypoScript осуществляется различными приложениями-анализаторами в TYPO3, например Анализатор шаблона для Шаблонов TypoScript или модуль Управление пользователями или функция Tsconfig страниц в модуле Инфо. Они обычно позволяют просмотреть TypoScript в каждом контексте, подсвечивая синтаксис.

Вот пример из Tsconfig страницы:



В расширении "extdeveval" можно найти инструмент, "Code highlighting", который может проанализировать код TypoScript в этом случае:



Существуют различные режимы представления:

Аналитический режим (представлен выше) расцвечивает все части синтаксиса:

- Объекты и свойства приобретают светло-голубой фон (1)
- Значения – желтый фон (2)
- Пробелы имеют оливково зеленый фон (3)
- Операторам дается коричневый фон.

Другой режим "Block mode" (Блочный режим), просто идентифицирует код TypoScript:

И наконец Вас предупредят о найденных синтаксических ошибках, а игнорируемые данные будут подсвечены зеленым:

```
asdf = qwerty
} - ERROR: Line 1: An end brace is in excess.
asdf { d
zxcvbnm uiop - ERROR: Line 3: Object Name String, "zxcvbnm" was not preceeded by any operator, =<>({
backgroundColor = blue
backgroundColor. {
 # This is a comment
transparency = 95%
another_property = 123
another_property2.first = 456
another_property2 {
second = 44
} asdf
} - ERROR: Line 13: The script is short of 1 end brace(s)
```

Отладка

Отладка TypoScript на предмет ошибок синтаксиса делается при помощи этого инструмента или других, использующих подсветку синтаксиса. Но это может предупредить о *синтаксических* ошибках, в то же время, комбинация объектов и свойств зависит от контекста и не может быть выявлено анализатором TypoScript.

На данный момент не существует инструментов проверки семантики для Шаблонов TypoScript, Tsconfig страниц или TSconfig пользователей (написано летом 2003)

(Этот инструмент подсветки синтаксиса доступен начиная с версии TYPO3 3.6.0)

Мифы, ЧаВо и признания

Эта страница – заключительно слово во вступительной главе синтаксиса TypoScript. Здесь имеется несколько замечаний, ответов на все еще имеющиеся у Вас вопросы. Итак, приступим:

Миф: "TypoScript – язык сценариев"

Это заблуждение, говорить, что TypoScript похож на PHP или JavaScript. Из предыдущих страниц Вы узнали, что TypoScript, строго говоря – только синтаксис. Но, применение синтаксиса TypoScript для создания Шаблонов TypoScript похоже на программирование и, тем более, если провести параллели с XSLT.

В любом случае, TypoScript НЕ сопостовим с языком сценариев, подобных PHP или JavaScript. Фактически, если TYPO3 и предлагает язык сценариев, то это непосредственно PHP! TypoScript только лишь API, часто используемая для настройки базового кода PHP.

Подытожим; само по себе название "TypoScript" вводит в заблуждение. Мы сожалеем об этом, но уже поздно что-либо менять.

Миф: "У TypoScript тот же синтаксис, что и у JavaScript"

TypoScript разрабатывался для упрощения работы и использования. Поэтому и синтаксис, в какой-то мере похож на JavaScript. Но повторюсь; очень опасно это говорить, все, что касается синтаксиса – TypoScript не процедурный язык программирования!

Миф: "TypoScript – частный стандарт"

Так как TypoScript не язык сценариев, то его ни в коем случае нельзя сравнивать с PHP, JavaScript, Java или какими-либо *языками сценариев*.

Однако, по сравнению с XML или массивами PHP (также содержащими *информацию*) можно сказать, что TypoScript – частный синтаксис, так как массив PHP или файл XML можно использовать для хранения той же информации, что и TypoScript. Но это *не* является недостатком: для хранения и обмена *содержимым,* TYPO3 использует SQL (или XML если хотите), для хранения *значений настроек* XML не подходит – с TypoScript проще работать (читайте дальше).

Утверждать, что TypoScript еще один стандарт, как аргумент против TYPO3, действительно несправедливо, так как TypoScript сразу представляется новым языком программирования или чем-то подобным. Да, у TypoScript свой *синтаксис,* но очень удобный, и, приобретя навыки работы, Вы найдете его очень удобным. В остальных случаях TYPO3 использует официальные стандарты, например PHP, SQL, XML, XHTML и т.п. для хранения и обработки *внешних* данных.

Наиболее широко TypoScript используется с Записями Шаблонов TypoScript. Понятно, что TypoScript выглядит сложным, когда Вы видите, сколько настроек движка Внешнего интерфейса можно производить через записи Шаблонов TypoScript. Но TypoScript всего лишь API (интерфейс) для базовой функциональности PHP. И говоря об огромном количестве предоставляемых им функция, подумайте, а лучше было бы непосредственно использовать еще больший функционал PHP?! При этом понадобиться прочесть даже больше документации по API для ее понимания, кроме того у Вас не будет довольно обтекаемой абстракции Шаблонов TypoScript. Поэтому нужно сказать: без TypoScript на изучение всего богатства функционала ушло бы гораздо больше времени!

Миф: "TypoScript очень сложный"

TypoScript прост на самом деле. Конечно, он станет сложным и "выйдет из под контроля" при возрастании объема кода! Но это решается:

- Дисциплинированное кодирование: организуйте свой TypoScript так, чтобы визуально во всем разбираться.
- Используйте подстветку синтаксиса для очистки своего кода, и как резюме сделанной работы в том числе.

А почему вместо этого не использовать XML?

Несколько раз TypoScript сравнивался с XML, ведь оба "языка" всего лишь оболочки (frameworks) для хранения информации. Кроме того, что XML – стандарт W3C (в отличии от TypoScript.. :-), основная разница в том, что XML лучше использовать для большого количества информации, в которой нужна высокая степень "точности", в то время как TypoScript лучше подходит для неболшой "специальной" информации — например значения настроек.

Фактически, структура данных, описываемая на TypoScript может быть описана и на XML. В настоящий момент, Вы *не* можете использовать XML как альтенативу TypoScript (написано в июле 2003), но это может произойти. Позвольте привести фиктивный пример того, как структура на TypoScript может быть представлена в "TSML" (наше условное название для несуществующего языка TypoScript-Mark Up Language):

```
styles.content.bulletlist = TEXT
styles.content.bulletlist {
  current = 1
  trim = 1
  if.isTrue.current = 1
        # Копирование объекта "styles.content.parseFunc" в это место
  parseFunc < styles.content.parseFunc
  split {
    token.char = 10
    cObjNum = 1
    1.current < .cObjNum
    1.wrap = <li>
  }
    # Помещение в теги:
  fontTag = <ol type="1"> | </ol>
  textStyle.altWrap = {$styles.content.bulletlist.altWrap}
}
```

Заняло 17 строк TypoScript кода, теперь переведем эту информацию в XML структуру:

```
<TSML syntax="3">
    <styles>
        <content>
            <bulletlist>
                TEXT
                <current>1</current>
                <trim>1</trim>
                <if>
                    <isTrue>
                        <current>1</current>
                    </isTrue>
                </if>
                  <!-- Копирование объекта "styles.content.parseFunc" в это место -->
                <parseFunc copy="styles.content.parseFunc"/>
                <split>
                    <token>
                        <char>10</char>
                    </token>
                    <cObjNum>1</cObjNum>
                    <num:1>
                        <current>1</current>
                        <wrap>&lt;li&gt;</wrap>
                    </num:1>
                </split>
                  <!-- Помещение в теги: -->
                <fontTag>&lt;ol type=&quot;1&quot;&gt; | &lt;/ol&gt;</fontTag>
                <textStyle>
                    <altWrap>{$styles.content.bulletlist.altWrap}</altWrap>
                </textStyle>
            </bulletlist>
        </content>
    </styles>
</TSML>
```

Заняло строки XML – в два раза больше! А в байтах значительно больше. Отличная демонстрация того *почему не XML*! XML просто не подходит для того, с чем отлично справляется TypoScript. По крайней мере, Вы можете на этом примере понять, что из себя представляет TypoScript в сравнении с XML.

Разумно использовать XML как альтернативу TypoScript, только по причине существования редакторов XML, которые могут легко преобразовать введенные XML данные в подобную приведенной структуру.

API синтаксического анализатора TypoScript

Вступление

Если Вы хотите использовать TypoScript в своих приложениях TYPO3, это очень просто. Синтаксический анализатор TypoScript доступен для Вас, за исключением одной вещи, требующей усилий – установки PHP.

Этот раздел в основном предназначен для обучения тому, как можно перевести свои строки TypoScript в структуру массива PHP. Это упражнение даже может помочь лучше понять прямой характер TypoScript.

Помните, что следующие страницы предназначены для опытных разработчиков TYPO3 и требуют хорошего знания PHP.

Интерпретация пользовательского TypoScript

Давайте предположим, что Вы создали приложение к TYPO3, например расширение. Вы определили основные параметры, редактируемые непосредственно в полях формы элемента содержимого расширения. Но Вы хотите, чтобы опытные пользователи могли бы делать детальную настройку, но вместо добавления массы таких параметров в интерфейс, – что загромоздило бы его, Вы скорее хотите, чтобы у опытных пользователей было текстовое поле, в которое они могли бы вводить закодированную информацию о настройке, описанную в маленькой справке.

Справка может выглядеть так:

Основной уровень

Свойство	Тип данных	Описание	По умолчанию
colors	->COLORS	Определяет цвет различных элементов.	
adminInfo	->ADMINFO	Определяет контактную информацию для администраторов cc-emails	
headerImage	Ссылка на файл	Ссылка на файл изображения, относительно пути сайта (PATH_site)	

->COLORS

Свойство	Тип данных	Описание	По умолчанию
backgroundColor	HTML-цвет	Цвет фона чего-либо...	white
fontColor	HTML-цвет	Цвет текста чего-либо...	black
popUpColor	HTML-цвет	Цвет тени чего-либо...	#333333

->ADMINFO

Свойство	Тип данных	Описание	По умолчанию
cc_email	строка	Адрес email чего-либо...	
cc_name	строка	Название чего-либо...	
cc_return_adr	строка	Адрес отвена на чего-либо...	[servers]
html_emails	Булева переменная	Если установлено, email в виде HTML	false

Так здесь есть "объекты" и "свойства" которые Вы предлагаете настроить опытным пользователям дополнения. Эта справка говорит, *какую информацию имеет смысл* помещать в поле TypoScript (семантика), потому что Вы запрограммировали свое приложение на использование этой информации.

A case story

Теперь представим, что пользователь ввел следующую настройку TypoScript в предложенное Вами поле настройки:

```
colors {
  backgroundColor = red
  fontColor = blue
}
adminInfo {
  cc_email = email@email.com
  cc_name = Copy Name
}
showAll = true

[UserIpRange = 123.456.*.*]
```

```
  headerImage = fileadmin/img1.jpg

[ELSE]

  headerImage = fileadmin/img2.jpg

[GLOBAL]

  // Wonder if this works... :-)
wakeMeUp = 7:00
```

В версии с подсветкой синтаксиса вроде этой (ниже), видно, что *нет синтаксических ошибок* и в этом отношении все отлично:

```
 0: colors {
 1:   backgroundColor = red
 2:   fontColor = blue
 3: }
 4: adminInfo {
 5:   cc_email = email@email.com
 6:   cc_name = Copy Name
 7: }
 8: showAll = true
 9:
10: [UserIpRange = 123.456.*.*]
11:
12:   headerImage = fileadmin/img1.jpg
13:
14: [ELSE]
15:
16:   headerImage = fileadmin/img2.jpg
17:
18: [GLOBAL]
19:
20:   // Wonder if this works... :-)
21: wakeMeUp = 7:00
```

(Подсвтеку синтаксиса TS (XML и PHP) можно сделать при помощи расширения “extdeveval”).

Для анализа этого TypoScript можно использовать следующий код, при условии, что переменная $tsString содержит приведенный выше код TypoScript:

```
 3: require_once(PATH_t3lib.'class.t3lib_tsparser.php');
 4:
 5: $TSparserObject = t3lib_div::makeInstance('t3lib_tsparser');
 6: $TSparserObject->parse($tsString);
 7:
 8: echo '<pre>';
 9: print_r($TSparserObject->setup);
10: echo '</pre>';
```

- Строка 3: включение класса анализатора TypoScript (скорее всего уже сделано и во внешнем, и во внутреннем интерфейсе TYPO3)
- Строка 5: Создание объекта класса анализатора.
- Строка 6: запуск анализа содержимого TypoScript в $tsString.
- Строка 8-10: вывод результата анализа в $TSparserObject->setup

Результат выполнения этого кода:

```
Array
(
    [colors.] => Array
        (
            [backgroundColor] => red
            [fontColor] => blue
        )

    [adminInfo.] => Array
        (
            [cc_email] => email@email.com
            [cc_name] => Copy Name
        )

    [showAll] => true
    [headerImage] => fileadmin/img2.jpg
    [wakeMeUp] => 7:00
)
```

Теперь Ваше приложение может использовать эту информацию, например так:

```
echo '<table bgcolor="'.$TSparserObject->setup['colors.']['backgroundColor'].'">
    <tr>
        <td>
            <font color="'.$TSparserObject->setup['colors.']['fontColor'].'">HELLO WORLD!</font>
        </td>
    </tr>
</table>';
```

Как видите, некоторые из свойств TypoScript (или *пути объектов*) из справки здесь представлены. Здесь нет никакой мистики, и фактически так используется весь TypoScript в определенном контексте; TypoScript – просто значения для соответствующей настройки действий основного кода PHP — параметры, аргументы функций; TypoScript – это API для инструктирования основной системы.

Это значит, что сейчас мы можем начать обоснованно говорить о неправильно информации в TypoScript — понятно, что два введенных в TypoScript параметра не имеют никакого смысла: “showAll” и “wakeMeUp”. Оба свойства не определены в справочной таблице, а, значит, не реализованы в коде PHP. Тем не менее, синтаксический анализатор не обнаружил ошибок, так как использовался правильный синтаксис для определения этих свойств. Другое дело, что они не нужны, тоже, что определить в PHP переменную и никогда ее не использовать! Потеря времени и, возможно, головная боль в будующем.

Как было отмечено, нет никакой "проверки семантики" для любого TypoScript приложения при такой записи. Хотя это было бы очень полезно, предупреждать об использовании несуществующих свойств (хотя это могуть быть лишь орфографические ошибки).

Реализация условий

Теперь мы знаем, как анализируется TypoScript и нам нужна лишь одна вещь – реализация выполнения условий. Как говорилось несколько раз, *выполнение* условий проверятся вне TypoScript и все, что нужно сделать — сообщить внешнему процессу об условиях и передать объект, как второй параметр в функцию анализа. Вот, как это сделано:

```
 1: require_once(PATH_t3lib.'class.t3lib_tsparser.php');
 2:
 3: class myConditions {
 4:     function match($conditionLine)    {
 5:         if ($conditionLine == '[TYPO3 IS GREAT]')    return true;
 6:     }
 7: }
 8: $matchObj = t3lib_div::makeInstance('myConditions');
 9:
10: $TSparserObject = t3lib_div::makeInstance('t3lib_tsparser');
11: $TSparserObject->parse($tsString, $matchObj);
12:
13: debug($TSparserObject->setup);
```

Некоторые замечания по листингу:

- Строки 3-8 определяют очень простой класс, содержащий функцию match().
  Функция "match()" должна существовать, и брать строку, как параметр, возвращая булево значение. Эта функция должна сравнивать строку с определенными Вами значениями.
  Здесь, если сравниваемая строка содержить значение «[TYPO3 IS GREAT]», то возвращает истину и анализирует следующий TypoScript.
- Строка 11: здесь экземпляр объекта $matchObj, класса “myConditions” передается в анализатор.
- Строка 13: небольшое побочное замечание: вместо использования PHP функции “print_r” используем классическую для TYPO3 функцию debug(), распечатывающую массив в таблицу HTML – некоторые из нас думают, что это хороший способ представить содержимое массива (составьте свое мнение на основе приводимых ниже рисунков).

В любом случае, давайте оценим класс условия приведенного выше кода, на основе анализа кода TypoScript:

```
 0: someOtherTS = 123
 1:
 2: [TYPO3 IS GREAT]
 3:
 4: message = Yes
 5: someOtherTS = 987
 6:
 7: [ELSE]
 8:
 9: message = No
10:
11: [GLOBAL]
12:
13: someTotallyOtherTS = 456
```

Из этого листинга мы ожидаем, что путь объекта "message" будет "Yes", если «[TYPO3 IS GREAT]» соответствует критериям и возвратит истину. Посмотрим:

someOtherTS	987
message	Yes
someTotallyOtherTS	456

Согласно этому результату, все работает!

Давайте попытаемся изменить строку 2:

```
1:
2: [TYPO3 IS great]
3:
```

Вот результат анализа:

someOtherTS	123
message	No
someTotallyOtherTS	456

Как видите, значение "message" теперь "No", значит условия не выполнены; Строка «[TYPO3 IS great]», *не* тоже самое, что «[TYPO3 IS GREAT]»! Значение "someOtherTS" также изменено на "123", т.е. значение, установленное до условия, при выполнении которого оно меняется, чего не случилось.

Реальный пример

Вероятно, Вы не хотите оценивать условия по простой строке. Скорее всего, Вы хотели бы создать правила для синтаксиса, а затем проанализировать строку условия. Один пример может стать такого класса измененным условием, замеченным в распечатках TypoScript предыдущих разделов, «[UserIpRange = 123.456.*.*]»:

```
 1: class myConditions {
 2:     function match($conditionLine)    {
 3:             // Получение значения внутри квадратных скобок:
 4:         $insideSqrBrackets = trim(ereg_replace('\]$','',substr($conditionLine,1)));
 5:
 6:             // Разбитие значения на ключ и значение, по знаку "="
 7:         list($key,$value) = explode('=',$insideSqrBrackets,2);
 8:
 9:         switch(trim($key))    {
10:             case 'UserIpRange':
11:       return t3lib_div::cmpIP(t3lib_div::getIndpEnv('REMOTE_ADDR'), trim($value)) ? TRUE : FALSE;
12:             break;
13:             case 'Browser':
14:                 return $GLOBALS['CLIENT']['BROWSER']==trim($value);
15:             break;
16:         }
17:     }
18: }
```

Этот класс работает примерно так:

- Строка 4: удаляются квадратные скобки в начале (а возможно и в конце) строки условия.
- Строка 7: строка условия без квадратных скобок разбирается на ключ и значение, разделяемое знаком «=»; мы пытаемся реализовать концепцию оценки источника данных по значению.
- Строки 9-16: эта конструкция-переключатель позволяет "key" быть либо "UserIpRange", либо "Browser" (указатель источника данных) и соответственно трактовать значение после знака равенства.

Давайте попытаемся разобрать TypoScript листинг из предыдущего раздела:

```
 0: colors {
 1:   backgroundColor = red
 2:   fontColor = blue
 3: }
 4: adminInfo {
 5:   cc_email = email@email.com
 6:   cc_name = Copy Name
 7: }
 8: showAll = true
 9:
10: [UserIpRange = 123.456.*.*]
11:
12:   headerImage = fileadmin/img1.jpg
13:
14: [ELSE]
15:
16:   headerImage = fileadmin/img2.jpg
17:
18: [GLOBAL]
19:
20:   // Wonder if this works... :-)
21: wakeMeUp = 7:00
```

Результат анализа можно представить следующим образом:

colors.	backgroundColor	red
	fontColor	blue
adminInfo.	cc_email	email@email.com
	cc_name	Copy Name
showAll	true	
headerImage	fileadmin/img2.jpg	
wakeMeUp	7:00	

Как можно заметить значение свойства "headerImage" взято из раздела условия [ELSE], таким образом «[UserIpRange = 123.456.*.*]» не соответствует условиям – что не удивительно, так как ни у кого не может быть IP адрес в диапазоне "123.456.*.*" !

Давайте изменим строку 10 в TypoScript:

```
 9:
10: [UserIpRange = 192.168.*.*]
11:
```

Так как я в настоящий момент нахожусь во внутренней сети с диапазоном IP адресов, соответствующих условиям, при анализе TypoScript, получаем:

colors.	backgroundColor	red
	fontColor	blue
adminInfo.	cc_email	email@email.com
	cc_name	Copy Name
showAll	true	
headerImage	fileadmin/img1.jpg	
wakeMeUp	7:00	

... так и есть!

Реализация условий II

А как насчет реализации комбинации условий? Например, в контексте Шаблонов TypoScript можно поместить несколько "условий" в одно (настоящее) условие:

```
[browser = netscape][browser = opera]
someTypoScript = 123
[GLOBAL]
```

Условия оцениваются по результирующей ИЛИ комбинации всех подусловий (реализовано в классе t3lib_matchCondition). Можно реализовать что-то подобное, и даже лучше. Например подобный синаксис:

```
[ CON 1 ] && [ CON 2 ] || [ CON 3 ]
```

Это читается так: "Истина, если истинны условия 1 и 2 ИЛИ истинно условие 3". Другими словами, мы реализуем как И, так и ИЛИ условия.

Это реализуется следующим образом:

```
  1: class myConditions {
  2:
  3:     /**
  4:      * Разделение входящей строки условий на И и ИЛИ части.
  5:      * Которые оцениваются отдельно и окончательно комбинируются.
  6:      */
  7:     function match($conditionLine)    {
  8:             // Получение значений внутри квадратных скобок
  9:             // строки условий:
 10:         $insideSqrBrackets = trim(ereg_replace('\]$','',substr($conditionLine,1)));
 11:
 12:             // "пропуск" оператора, приоритет ИЛИ:
 13:         $ORparts = split('\][[:space:]]*\|\|[[:space:]]*\[',$insideSqrBrackets);
 14:         foreach($ORparts as $andString)    {
 15:             $resBool = FALSE;
 16:
 17:                 // Разделение по "&&" и оператору:
 18:             $ANDparts = split('\][[:space:]]*\&\&[[:space:]]*\[',$andString);
 19:             foreach($ANDparts as $condStr)    {
 20:                 $resBool = $this->evalConditionStr($condStr) ? TRUE : FALSE;
 21:                 if (!$resBool)    break;
 22:             }
 23:
 24:             if ($resBool)    break;
 25:         }
 26:         return $resBool;
 27:     }
 28:
 29:     /**
 30:      * Оценка внутренней части условий.
 31:      */
 32:     function evalConditionStr($condStr)    {
 33:             // Разделение значения на ключ и значение по знаку "="
 34:         list($key,$value) = explode('=',$condStr,2);
 35:
 36:         switch(trim($key))    {
 37:             case 'UserIpRange':
                                                                                  38:
               return t3lib_div::cmpIP(t3lib_div::getIndpEnv('REMOTE_ADDR'), trim($value)) ? TRUE : FAL
SE;
 39:             break;
 40:             case 'Browser':
 41:                 return $GLOBALS['CLIENT']['BROWSER']==trim($value);
 42:             break;
 43:         }
 44:     }
 45: }
```

При этой реализации, строка условий будет похожа на эту:

```
  9:
 10: [UserIpRange = 192.168.*.*] && [Browser = msie]
 11:
 12:   headerImage = fileadmin/img1.jpg
 13:
```

Так, если я вхожу в правый диапазон IP И имею соответствующий браузер, значение "headerImage" будет "fileadmin/img1.jpg"

Если мы изменим TypoScript следующим образом, следуя тем же условиям, а браузер будет Netscape, то условия будут выполняться независимо от диапазона IP:

```
 9:
10: [UserIpRange = 192.168.*.*] && [Browser = msie] || [Browser = netscape]
11:
12:   headerImage = fileadmin/img1.jpg
```

Потому что условия читаются как показано со скобками:

("UserIpRange = 192.168.*.*» И "Browser = msie" **) ИЛИ** "Browser = netscape"

Теперь может быть пробематично употреблять операторы «||» и «&&». Например:

```
 9:
10: [UserIpRange = 192.168.*.*] || [UserIpRange = 212.237.*.*] && [Browser = msie]
11:
12:   headerImage = fileadmin/img1.jpg
```

Представьте эти условия: хотелось бы, чтобы это было так "если диапазон адресов IP пользователя попадает в 1 или 2 и в любом случае браузер должен быть MSIE!». Но сейчас это выглядит так: диапазон IP пользователя соответствует 192.168.... ИЛИ диапазон 212.237.... и браузер MSIE.

Формально это выглядит так:

("UserIpRange = 192.168.*.*» **ИЛИ** "UserIpRange = 212.237.*.*» **) И** "Browser = msie"

Я думаю, что это можно решить путем слияния условий ИЛИ – просто подразумевая оператор ИЛИ как "условие" при отсутствии оператора. При этом, ту же строку можно было бы записать так:

```
 9:
10: [UserIpRange = 192.168.*.*][UserIpRange = 212.237.*.*] && [Browser = msie]
11:
12:   headerImage = fileadmin/img1.jpg
```

Line 10 will be understood in this way:

```
[UserIpRange = 192.168.*.*](implied OR here!)[UserIpRange = 212.237.*.*] && [Browser = msie]
```

Функция match() для класса условий должна быть изменена следующим образом:

```
 1:     /**
 2:      * Разделение входящей строки условий на И и ИЛИ части.
 5:      * Которые оцениваются отдельно и окончательно комбинируются.
 4:      */
 5:     function match($conditionLine)    {
 6:             // Получение значений внутри квадратных скобок
 9:             // строки условий:
 8:         $insideSqrBrackets = trim(ereg_replace('\]$','',substr($conditionLine,1)));
 9:
10:             // "пропуск" оператора, приоритет ИЛИ:
11:         $ORparts = split('\][[:space:]]*\|\|[[:space:]]*\[',$insideSqrBrackets);
12:         foreach($ORparts as $andString)    {
13:             $resBool = FALSE;
14:
15:                 // Разделение по "&&" и оператору:
16:             $ANDparts = split('\][[:space:]]*\&\&[[:space:]]*\[',$andString);
17:             foreach($ANDparts as $subOrStr)    {
18:
19:                     // Разделение при отсутствии оператора между ] и [ (подразумеваетя ИЛИ)
20:                 $subORparts = split('\][[:space:]]*\[',$subOrStr);
21:                 $resBool = FALSE;
22:                 foreach($subORparts as $condStr)    {
23:                     if ($this->evalConditionStr($condStr))    {
24:                         $resBool = TRUE;
25:                         break;
26:                     }
27:                 }
28:
29:                 if (!$resBool)    break;
30:             }
31:
32:             if ($resBool)    break;
33:         }
34:         return $resBool;
35:     }
```

Именно так.

Фактически, эту функию для Шаблонов TypoScript можно реализовать один раз. Сейчас предлагается лишь разделение условий с ИЛИ но можно применить и пердставленный анализ. Как знать, может это уже и будет реализовано, пока вы читаете этот документ...

Приложение относительного нашего дополнения

Помните предыдущий раздел? Мы определили три таблицы со свойствами, которые могут использоваться в TypoScript в контексте нашего приложения. К этой справке теперь мы должны добавить раздел с условиями:

1: Синтаксис строки:

Условия разбиваются на маленькие часи с логикой И и ИЛИ. Каждая подчасть условий в строке разделена по «] (оператор) [», где оператор это «&&» (И) , «||» (ИЛИ) или может отсутствовать (что значит ИЛИ "после" И).

Таким образом, формат строки условий такой:

[COND1] || [COND2] && [COND3] [COND4] ....и т.д.

Где соблюдается следующий порядок операторов:

[COND1] || **(** [COND2] && **(** [COND3] [COND4] **))**

(Примечание: пустое пространство между COND3 и COND4, неявно подразумевается ИЛИ)

2: Синтаксис подчастей:

Для каждой подчасти (например «[COND 1]») содержимое оценивается следующим образом:

[KEY = VALUE]

где KEY обозначает тип условия из этой таблицы:

KEY	Описание	Пример
UserIpRange	Истинно, если IP адрес клиента соответствует заданному образцу. Значение сравнивается с REMOTE_ADDR функцией t3lib_div::cmpIP(), из которой можно узнать синтаксис.	[UserIpRange = 192.168.*.*]
Browser	Истинно, если браузер клиента соответствует одному из ключевых слов: Вы можете использовать следующие значения: **konqu** = Konqueror **opera** = Opera **msie** = Microsoft Internet Explorer **net** = Netscape (или любой другой) Значение сравнивается функцией t3lib_div::clientInfo(), из которой можно узнать о значениях для браузера.	[Browser = msie]