

Real URL

Ключ расширения: **realurl**

Авторские права 2003-2005

`martin@beryllium.net`, `<martin@beryllium.net>`

&

`Kasper Skårhøj <kasperYYYY@typo3.com>`

&

`Дмитрий Дулепов <dmitry@typo3.org>`

Документ публикуется под лицензией открытого информационного содержимого
доступной на <http://www.opencontent.org/opl.shtml>

Содержимое документа относится к TYPO3
- GNU/GPL CMS/Framework доступной на www.typo3.com

Содержание

Real URL	1	URL.....	19
Введение	1	class.tx_realurl_advanced.php	20
Предназначение.....	1	Введение.....	20
Функциональность.....	2	Настройка.....	20
Настройка	2	Автоматическая настройка	22
Установка.....	2	Обзор.....	22
Кодирование/декодирование URL в фоне.....	3	Автоматическая настройка расширения.....	22
Инструкции настройки.....	5	Приложение	23
Обработка относительных ссылок при понятных		Список ToDo	24
		Список изменений	24

Введение

Предназначение

Расширение автоматически преобразует URL с параметрами GET во внешнем пользовательском интерфейсе (например `index.php?id=123&type=0&L=1`) в виртуальный путь, так называемый "Понятный URL" (например `dutch/contact/company-info/page.html`) и обратно. Объективно, такие URL должны быть максимально понятны человеку.

Расширение очень гибко и может делать простую трансформацию ID страниц в кодировку почти любых возможных комбинаций GET параметров.

Примеры

Понятный URL	TYP03 id и тип
<code>http://www.domain.com/</code>	<code>id=0, type=0</code>
<code>http://www.domain.com/products/product1/features/</code>	<code>id=123, type=0</code>
<code>http://www.domain.com/products/product1/features/leftframe.html</code>	<code>id=123, type=2</code>

Фон

TYPO3 работает с ID страниц. Это превосходно функционирует, несмотря на то, что URL выглядят очень громоздко (“...index.php?id=123&type=0&L=1...” и т.д.). Есть приемы (simulateStaticDocuments), но это фальшивка: идентификатор все еще должен быть представлен в URL, что не желательно. Кроме того, будут показаны лишь заголовки страницы, а не полный путь (или строка к корню) к странице.

Обычно Вы набираете путь и название документа, но TYPO3 работает исключительно с идентификаторами страниц. Расширение RealURL предоставляет путь к переводу идентификаторов страниц в читаемые и легко запоминающиеся (виртуальные) URL.

Для расширения требуется модуль Apache “mod_rewrite” для перезаписи виртуальных URL сайта в движке внешнего интерфейса TYPO3.

Обычно оно работает сразу после установки, но вы должны сделать обзор всех ссылок на HTML странице, на предмет, являются ли они абсолютными URL или установлены относительные теги. Оба метода имеют преимущества и недостатки, но возможно Вам придется подправить Ваш шаблон/код для совместимости.

Функциональность

- Поддерживаются различные схемы кодирования пути к странице, включая определяемые пользователем
 - Название страницы может содержать пробелы и символы /,.&@ , URL все равно будет выглядеть приемлемо.
 - URL генерируются в нижнем регистре, для привлекательного вида
 - При переименовании страницы, старый URL все равно может использоваться (смотрите дальше в руководстве пользователя), таким образом, страница, индексированная, например Google, будет найдена.
- Предлагается дополнительный перевод почти всех наборов параметров GET в/из виртуального URL
 - Перевод между строкой запроса GET (“...&tx_myext[blabla]=123&type=2...”) и виртуальным URL (“.../123/2”) прозрачно для TYPO3 и всех расширений; Единственное требование, использование внутренней функции TYPO3 для формирования ссылок (“t3lib_cObj::typolink”, “t3lib_template::linkData”)
- URL буферизуются, таким образом, перевод между URL и ID осуществляется моментально.
- Возможна обработка различных структур или других типов страниц
- URL многоязычны: если Вы просматриваете на русском, Вы увидите русские URL
- Однажды настроенная система работает полностью автоматически, создавая и обновляя существующие URL
- Вы легко можете видеть, куда указывают ссылки, так как формируется «целевой» URL , а не URL к ссылке.
- Автоматически обрабатываются установки TYPO3 в папки, отличающиеся от корня веб сайта
- Не обрабатываются URL остающихся параметров названий переменных GET
- Несовместимо с “simulateStaticDocuments” .

Настройка

Установка

Установка расширения проходит в четыре шага:

1. Установка из Менеджера расширений
2. Настройка Apache / .htaccess
3. Изменение записей шаблона TypoScript настройкой под Real URL
4. Настройка расширения в typo3conf/localconf.php

Установка расширения

Это довольно хорошо освещено в документации по TYPO3: нажать на маленький серый кружок со значком плюс и согласится со всеми вносимыми изменениями. Это все, что нужно сделать.

Настройка Apache

RealURL работает, формируя «виртуальные пути» к «виртуальным файлам». Они физически не существуют, поэтому нужно, чтобы Apache позволил программе PHP обработать запрос, если файл найти невозможно. Таким образом, все URL на страницы (например www.server.com/products/product1/left.html) будут перенаправлены на /index.php, которая переведет URL в параметры GET. Настоящие файлы (например рисунки, внутренний интерфейс TYPO3, статические html файлы и др.) будут все также обрабатываться через Apache.

Нужно поместить предоставленный файл .htaccess (называемый _htaccess) в корень установленного TYPO3.

Либо, Вы можете включить в **httpd.conf** следующие строки, возможно в раздел **VirtualHost**. Пример:

```
<VirtualHost 127.0.0.1>
```

```

DocumentRoot /var/www/typo3/dev/testsite-3/
ServerName www.test1.intra

RewriteEngine On
RewriteRule ^/typo3$ - [L]
RewriteRule ^/typo3/.*$ - [L]

RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-l
RewriteRule .* /index.php
</VirtualHost>

```

ВНИМАНИЕ: ИСПОЛЬЗОВАНИЕ httpd.conf, как известно на данный момент, не работает, так как не корректно t3lib_div::getIndpEnv('TYPO3_SITE_URL'). Решение до сих пор ищется.

Если Вы помещаете их в файл **.htaccess**, нужно их немного изменить, удалив начальные слешы (“/”):

```

RewriteEngine On
RewriteRule ^typo3$ - [L]
RewriteRule ^typo3/.*$ - [L]

RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-l
RewriteRule .* index.php

```

Что заставит Apache перезаписать любой URL , не являющийся именем файла, директории или симлинком. И оставит без изменения адреса, начинающиеся с /typo3/.

Примечание: для этого нужен модуль Apache “mod_rewrite”!

Также просмотрите дополнительную информацию по mod_rewrite.

Настройка TYPOScript

Как и в “simulateStaticDocuments” , нужно задействовать формирование названий виртуальных фалов/путей в записи TYPOScript , иначе Ваш веб сайт не сможет использовать новый метод кодировки URL.

Но это просто; всего лишь поместите эти три строки в запись TYPOScript для основного шаблона Вашего сайта:

```

0: config.simulateStaticDocuments = 0
1: config.baseURL = 1
2: config.tx_realurl_enable = 1

```

Строка 0 просто отключает “simulateStaticDocuments”, так как “realurl” с ним несовместим и просто не будет работать. Эта строка просто напомнит Вам об этом.

Строка 1 заставляет выводить во внешнем интерфейсе “<base>” тег в заголовке страниц. Это нужно ввиду того, что относительные ссылки на изображения, таблицы стилей и т.п. не смогут использоваться с виртуальными путями, если не сделать эту установку. Ниже Вы найдете подробное объяснение необходимости этого шага.

Строка 2 включает кодирование URL как виртуальных путей “понятных URL”.

Настройка расширений

Наконец, Вы, возможно, захотите настроить свойства для кодирования URL. Для простого использования настроек по умолчанию достаточно.

Настройки производятся в файле “localconf.php” в переменной \$TYPO3_CONF_VARS[‘EXTCONF’][‘realurl’]

Параметры приведены в соответствующем разделе, там же предлагается множество примеров.

Кодирование/декодирование URL в фоне

В этом разделе приводятся сведения о том, как происходит фоновая кодировка и расшифровка URL.

Главный принцип — кодирование и расшифровка должны быть полностью прозрачны для системы. Это значит, что с “realurl” будут работать любые расширения, формирующие ссылки методами самой TYPO3. Протестировать это Вы можете, используя “simulateStaticDocuments” - если работает с ним, то (скорее всего) будет работать и с “realurl”.

Реализована эта прозрачность посредством кодирования виртуального URL строго на основе параметров GET для методов кодера. И поступающий HTTP запрос к виртуальному URL, расшифровывается в набор GET параметров, записываемых обратно в глобальные переменные HTTP_GET_VARS / _GET, таким образом, любые приложения в системе, видят параметры, как истинные параметры GET.

Кодирование:

URL с параметрами GET -> понятный URL -> HTML страница

Кодирование URL производится, используя обработчик в методе t3lib_tstemplate::linkData(). Что настраивается в “realurl/ext_localconf.php”:

```
$TYPO3_CONF_VARS['SC_OPTIONS']['t3lib/class.t3lib_tstemplate.php']['linkData-PostProc'][] =
'EXT:realurl/class.tx_realurl.php:&tx_realurl->encodeSpURL';
```

Расшифровка:

HTTP запрос к понятному URL -> URL расшифровывается, перезаписывая значения HTTP_GET_VARS -> страница формируется как обычно

Расшифровка URL производится через обработчик в `tslib_fe::checkAlternativeIdMethods()`. И настраивается примерно так:

```
$TYPO3_CONF_VARS['SC_OPTIONS']['tslib/class.tslib_fe.php']['checkAlternativeIdMethods-PostProc'][] =
'EXT:realurl/class.tx_realurl.php:&tx_realurl->decodeSpURL';
```

Синтаксис “понятного URL”

Перед тем, как перейти к разделу настроек, важно понять, как виртуальный путь (понятный URL) расшифровывается системой. Давайте разберем следующий пример:

```
index.php?id=123&type=1&L=1&tx_mininews[mode]=1&tx_mininews[showUid]=456
```

В этом URL запрашивается страница с id 123 на языке “1” (датский) и с типом “1” (вероятно информационная структура из набора), и на этой странице отображается элемент мининюостей с id “456” в “моделе” меню мининюостей “1” (список). Эти параметры основного URL, могут быть переведены в понятный URL:

```
dk/123/news/list/456/page.html
```

Настройка “realurl” для этого превращения следующая:

```
1: $TYPO3_CONF_VARS['EXTCONF']['realurl']['_DEFAULT'] = array(
2:     'preVars' => array(
3:         array(
4:             'GETvar' => 'L',
5:             'valueMap' => array(
6:                 'dk' => '1',
7:             ),
8:             'noMatch' => 'bypass',
9:         ),
10:    ),
11:    'fileName' => array (
12:        'index' => array(
13:            'page.html' => array(
14:                'keyValues' => array (
15:                    'type' => 1,
16:                )
17:            ),
18:            '_DEFAULT' => array(
19:                'keyValues' => array(
20:                )
21:            ),
22:        ),
23:    ),
24:    'postVarSets' => array(
25:        '_DEFAULT' => array (
26:            'news' => array(
27:                array(
28:                    'GETvar' => 'tx_mininews[mode]',
29:                    'valueMap' => array(
30:                        'list' => 1,
31:                        'details' => 2,
32:                    )
33:                ),
34:                array(
35:                    'GETvar' => 'tx_mininews[showUid]',
36:                ),
37:            ),
38:        ),
39:    ),
40: );
```

Чтобы понять, как при этой настройке происходит перевод понятного URL обратно в параметры GET, давайте сначала посмотрим, как понятный URL разбивается на разделы. Общий синтаксис:

```
[TYPO3_SITE_URL] [preVars] [pagePath] [fixedPostVars] [postVarSets] [fileName]
```

Каждый из разделов (исключая `fileName`) может состоять из одного или нескольких *сегментов*, разделяемых “/”. Так, “news/list/456” - секция, состоящая из трех *сегментов*, “news”, “list” и “456”

Таким образом, приведенный понятный URL (<http://www.my-domain.dk/frontend/dk/123/news/list/456/page.html>), можно разбить на следующие секции:

Секция	Часть URL из примера	Основное описание	Комментарии по конфигурации
[TYPO3_SITE_URL]	http://www.my-domain.dk/frontend/	Эта часть URL - основной URL и место расположения основной программы "index.php" внешнего интерфейса – просто извлекается и не настраивается.	
[preVars]	dk/	Этот раздел может содержать от нуля до любого количества сегментов, разделенных "/". Каждый сегмент привязан к переменной GET в настройке с названием "preVars" (смотрите пример) Количество сегментов в секции pre-vars определяется массивом в настройке "preVars". Обычно, pre-var привязывается к параметру "L" GET, таким образом, язык сайта представлен первым сегментом виртуального пути.	Здесь, настраивается только одна pre-var, привязанная к GET переменной "L". Далее, таблица отображения говорит, что значению "dk" соответствует "1" в переменной GET при расшифровке. Также, если сегмент не "dk", он игнорируется. То есть, страница датской версии по умолчанию ".../dk/123/" - просто ".../123/"
[pagePath]	123/	Путь, определяющий ID страницы. По умолчанию, просто будет показан ID страницы, но можно показать и "contact/company_info/" через настройку ID страницы. Количество сегментов пути для перевода ID страницы, зависит от используемого метода.	В примере использован метод по умолчанию (без настройки), отображающий необработанный id страницы (или ее псевдоним); 123. Это слабый пример, без по-настоящему "понятного URL" ...
[fixedPostVars]		Фиксированные пост переменные, раздел фиксированных связей между переменными GET и сегментами пути, также как и pre-vars. Обычно их не используют для настройки целого сайта, так как общие параметры должны быть установлены как pre-vars, но Вы можете настроить фиксированные пост переменные для отдельной страницы, на которой находится определенное приложение. Обычно эта секция применяется для страницы с определенным ID, на которой работает специфическое системное приложение.	Не используется в этом примере.
[postVarSets]	news/list/456/	postVarSets разделы переменных GET, связанных (подобно pre-var) инициализацией первого сегмента пути, являющимся зарезервированным идентификационным словом для последовательности. Расшифровка postVarSets продолжается до тех пор, пока не будут переведены все упомянутые сегменты виртуального пути. Этот метод может использоваться для кодирования групп переменных GET (наборов), обычно используемых различными плагинами на веб сайте. Обычно postVarSets используются для настройки каждого используемого на веб сайте дополнения.	В этом примере используется одна переменная такого типа ("news/list/456/") где ключевое слово "news/" (первый сегмент) идентифицированное в следующий раздел ("list/456/") соответствуют переменным GET tx_mininews[mode] и tx_mininews[showUid] соответственно.
[fileName]	page.html	Имя файла всегда идентифицируется как сегмент виртуального пути после последнего слеша ("/"). При настройке "fileName", название файла может соответствовать ряду переменных GET, которые будут установлены, если имя файла будет соответствовать значению индексированного массива. Обычно используется при кодировке переменных GET "type" или "print".	В этом примере "type" со значением "1" соответствует виртуальному файлу "page.html". В любом другом случае, значение не будет установлено.

Инструкции настройки

Настройка "realurl" производится в массиве \$TYPO3_CONF_VARS['EXTCONF']['realurl'] который, опять же, содержит массивы. Инструкции настройки разделяются на таблицы, описывающие параметры, сгруппированные в массивы в пределах массива настроек.

Для того, чтобы Вы смогли понять эти параметры, прочитайте основную информацию, представленную ранее и взгляните на доступные здесь примеры.

\$TYPO3_CONF_VARS['EXTCONF']['realurl']

Ключ	Тип	Описание
[host-name]	->siteCfg или указатель на другой ключ ->siteCfg в том же массиве	Настройка кодировки понятных URL, на основе текущего имени хоста веб сайта. Возможна индивидуальная настройка для разных доменов одной базы данных. Если значение представлено строкой, система будет трактовать ее как указатель на другой ключ на этом же уровне и будет искать ->siteCfg. Имя хоста расположено в t3lib_div::getIndpEnv("TYPO3_HOST_ONLY") и всегда в нижнем регистре.
_DEFAULT	->siteCfg или указатель на другой ключ ->siteCfg в том же массиве	Configuration of default Speaking URL coding if no matches was found for the specific HOST name.

Пример

```
1: $TYPO3_CONF_VARS['EXTCONF']['realurl'] = array(
2:     '_DEFAULT' => array(
3:         ...
4:     ),
5:     'www.typo3.org' => array (
6:         ...
7:     ),
8:     'www.typo3.com' => 'www.typo3.org',
9:     'typo3.com' => 'www.typo3.org',
10:    '192.168.1.123' => '_DEFAULT',
11:    'localhost' => '_DEFAULT',
12: );
```

В этом примере, ключи “_DEFAULT” и “www.typo3.org” содержат, как предполагается, надлежащую конфигурацию “realurl”. Если имя хоста “www.typo3.com” или “typo3.com”, используется конфигурация “www.typo3.org”. Если имя хоста “192.168.1.123” или “localhost”, то используется конфигурация “_DEFAULT” (что избыточно, так как она устанавливается в любом другом случае!)

->siteCfg

Ключ	Тип	Описание
init	->init	Основная настройка расширения
redirects [path]	Переназначенный URL	Здесь Вы можете определить виртуальные пути, которые нельзя обработать через переменные GET, и легче перенаправить заголовок HTTP, как значение URL. Такое случается, если сценарий отправляет заголовок и завершается.
redirects_regex[regex]	Переназначенный URL	Расширенная версия настройки [redirects], где можно сопоставлять путь с regex. Можно использовать и обратные ссылки в перенаправляемом URL. Пример: <pre>'redirects_regex' => array('^downloads/(.*)' => 'ftp://dl.domain.tld/public/dl/\1',),</pre> В этой конфигурации, любой URL, начинающийся с “downloads/” будет перенаправлен на ftp-сервер. Например, “ http://www.domain.tld/downloads/software/game.exe ” будет перенаправлен на “ ftp://dl.domain.tld/public/dl/software/game.exe ” <pre>^francais/(.*)' => 'fr/\1',</pre> Этот пример перенаправляет любой, начинающийся с “francais/”, URL на “fr/”, что возможно является языковым ключом сайта.
preVars [0..x]	->partDef	Настройка pre-variables; фиксированный набор переменных связанных с начальными сегментами виртуального пути. Смотрите описание в предыдущем разделе этого документа.
pagePath	->pagePath	Настройка метода преобразования id в путь. Смотрите описание в предыдущем разделе этого документа.
fixedPostVars [pageIndex] [0..x]	->partDef	Настройка фиксированного набора post-variable не нуждающихся в ключевом слове для вызова интерпретатора. В основном, также как и pre-vars, но устанавливаются после pagePath. Смотрите описание в предыдущем разделе этого документа. Примечание: “pageIndex” позволяет определить индивидуальные страницы или все, используя слово “_DEFAULT”. Смотрите примечание после таблицы.
postVarSets [pageIndex] [keyword]	->postVarSet	Настройка набора post-variables; набор post-variables вызываемых по ключевому слову в виртуальном пути. Смотрите описание в предыдущем разделе этого документа. Примечание: “pageIndex” позволяет определить индивидуальные страницы или все, используя слово “_DEFAULT”. Смотрите примечание после таблицы. Важно: порядок postVarSets очень важен, будет выбрано первое ключевое слово, содержащее описание отдельной доступной переменной GET. Нужно стратегически упорядочить postVarSets.
fileName	->fileName	Настройка значения названия файла; названия файлов могут быть связаны со специальными значениями параметров GET. Смотрите описание в предыдущем разделе этого документа.

Примечание: в таблице определен ключевой параметр “pageIndex” для “fixedPostVars” и “postVarSets”. Он работает также, как указатель имени хоста, поскольку он сделан для верхнего уровня. Параметр может быть:

- либо id страниц, например “123”
- либо ключевое слово “_DEFAULT”.

Ключевое значение должно быть массивом (согласно определению выше), но, если это строка, то она интерпретируется как указатель на ключ этого же уровня.

Указатель не может быть установлен на “_DEFAULT”. Далее могут использоваться только id страниц (внутренние псевдономы страниц, установленные как параметры будут решаться сначала!).

Образец структуры

```

1: array(
2:   'init' => array(
3:     ...
4:   ),
5:   'redirects' => array(
6:     '' => 'cms/', // Если URL по умолчанию, перенаправить в поддиректорию "cms/"
7:     'test/' => 'http://www.test.test/', // Если поддиректория "test/", перенаправить на URL
8:     'myFolder/mySubfolder/myFile.html' => 'test/index.php',
9:   ),
10:  'preVars' => array(
11:    array(
12:      ...
13:    ),
14:    array(
15:      ...
16:    ),
17:  ),
18:  'pagePath' => array(
19:    ...
20:  ),
21:  'fixedPostVars' => array(
22:    '1383' => array (
23:      array(
24:        ...
25:      ),
26:      array(
27:        ...
28:      ),
29:    ),
30:    '123' => '1383'
31:  ),
32:  'postVarSets' => array(
33:    '_DEFAULT' => array (
34:      'consultancy' => array(
35:        ...
36:      ),
37:      'admin' => array(
38:        ...
39:      )
40:    ),
41:  ),
42:  'fileName' => array(
43:    ...
44:  )
45: );

```

Эта структура поможет Вам понять определенную в таблице выше структуру для уровня конфигурации “->siteCfg”. Обратите внимание, что пример для переназначения.

->init

Общая настройка расширения “realurl”

Ключ	Тип	Описание
doNotRawUrlEncodeParameterNames	булева переменная	Отменяет кодирование необработанных url не переведенных названий параметров GET во время кодирования. Что происходит: Во время кодирования понятного URL из параметров GET, любые параметры GET, которые невозможно перевести в понятный URL, будут возвращены. При этом, их названия останутся такими же, как в необработанном url, согласно полученным запросам. Это значит, что параметр, подобный “tx_myext[hello]=world” станет “tx_myext%5Bhello%5D=world”, что неопорно выглядит, но технически правильно.
enableCHashCache	булева переменная	Если установлено, параметр GET “сHash” сохраняется в буферной таблице, если это единственные параметры, оставленные как переменные GET. Это позволяет избавиться от параметров, остающихся от некоторых приложений, использующих буферизацию параметров.

Ключ	Тип	Описание
respectSimulateStaticURLs	булева переменная	Если установлено, все запросы, в которых путь понятного URL ведет к документу, и не содержит префикс (например "123.1.html"), не воспринимаются как понятные URL. Это гарантирует обратную совместимость со старыми URL, используемыми в simulateStaticDocuments на сайте. Эта установка игнорируется, если соблюдаются все эти условия: <ul style="list-style-type: none"> ● URL содержат промежуточные сегменты пути перед названием файла (т.е. выглядят как simulateStatic) ● ID страниц не находятся ядром TYPO3 до realurl ● установлен fileName.defaultToHTMLsuffixOnPrev
appendMissingSlash	булева переменная / строка	Если установлено, слеш будет добавлен в конец пути, если это не было установлено. Например, некоторые пишут "http://the.site.url/contact" без слеша в конце. При этом, "contact" рассматривается realurl как имя файла, в то время как пользователь имел ввиду название директории. Что исправляет эту проблему. Ключевое слово: "ifNotFile" Можно определить параметр "ifNotFile". При этом, слеш будет устанавливаться, только если последняя часть пути не похожа на название файла (т.е. не содержит символ точки ".").
adminJumpToBackend	булева переменная	Если установлено, в режиме "admin" не будет показана иконка редактирования во внешнем интерфейсе, перенаправляющая пользователя непосредственно в модуль страниц внутреннего интерфейса для редактирования страницы с текущим id.
postVarSet_failureMode	строка (ключи)	Ключевое слово: "redirect_goodUpperDir" . Составить URL из успешно отображенных частей и перенаправить туда. Ключевое слово: "ignore" : тихое принятие оставшихся частей. По умолчанию (пустое значение) страница 404 не найденная TYPO3 API внешнего интерфейса.
disableErrorLog	булева переменная	Если установлено, ошибки 404 не записываются в таблицу журнала.
enableUrlDecodeCache	булева переменная	Если установлено, включена расшифровка кешированных URL. Таблица кеша сбрасывается, при сбросе "всех кешей" в TYPO3. Элементы для расшифровки кеша по умолчанию доступны в течение 24 часов.
enableUrlEncodeCache	булева переменная	Если установлено, производится кеширование закодированных URL. Таблица кеша сбрасывается, при сбросе "всех кешей" в TYPO3.
emptyUrlReturnValue	строка	Если URL пуст, обычно подразумевается первая страница сайта. Значение этой строки дополнит пустой URL. Если установить значение true (PHP булева переменная, "TRUE"), будет возвращено значение baseURL, установленное в TSFE. Установка "." будет также работать как ссылка на корень сайта. Но это не столь красиво.

->partDef

Определение отображения сегмента виртуального пути и переменной GET, и наоборот.

Ключ	Тип	Описание
type	строка параметров	По умолчанию, массив, описывающий отображение переменных GET, но могут быть альтернативно настроены, установкой параметров: "action" : при установке, сегмент может определять различные действия, например, редактирование во внешнем интерфейсе ли перенаправление на определенный URL, установка параметров.
type = "action"		
index[segment] index["_DEFAULT"]	->actionConfig (или пустое значение, если нет действий)	Индексный массив, определяющий различные действия, первый сегмент пути используется как ключ для поиска действий в массиве. За деталями и примерами, обращайтесь к ->actionConfig.
Default type		
GETvar		Имя переменной GET для которой производится обработка. Требуется Значение GETvar преобразуется, согласно другим настройкам. В основном таким: <ul style="list-style-type: none"> ● Сначала проверяется позволена ли обработка "cond[prevValueInList]" и, если нет, возвращается. ● При совпадении, значение переводится посредством "valueMap". ● Если нет совпадений с "valueMap", производится сравнение с "noMatch". ● Если noMatch не вызывается, ищется поисковая таблица и, если определено, производится поисковый перевод ("lookUpTable"). ● Если не была определена поисковая таблица для перевода значения, ищется "valueDefault" и, если установлено, используется это значение. ● Если действий не совершается, передается необработанное значение.
cond[prevValueInList]	список значений	Если параметр установлен, сегмент обрабатывается, только если предыдущее значение находится в этом списке значений! Иначе обработка не производится.
valueMap	массив "сегмент" => "значение переменной Get" (таблица перевода)	ValueMap — статическая таблица перевода, где каждый параметр представляет собой сегмент понятного URL и истинное значение параметра. При кодировании URL, эта таблица изменяется, и, при дублировании значений, в качестве понятного URL используется последнее значение.
noMatch	строка параметров	Параметр, определяющий действие, если значение не соответствует значениям массива valueMap. "bypass" : если сегмент НЕ соответствует значениям "valueMap", он помещается в стек с возвратом/прерыванием (сохраняя параметр следующему сегменту) "null" : если значение НЕ найдено во valueMap, getParameter не устанавливается.
lookUpTable	->lookUpTable	Настройка таблицы базы данных, осуществляющей перевод id в строку псевдонима.
userFunc	userFunc	Функция пользователя, для перевода id<=>псевдоним. Используется при отсутствии "lookUpTable". Примерная конфигурация: <pre>'userFunc' => 'EXT:realurl/class.tx_realurl_userfunctest.php:&tx_realurl_userfunctest->main'</pre> Пример класса в "realurl" (class.tx_realurl_userfunctest.php) <pre>class tx_realurl_userfunctest { function main(\$params, \$ref) { if (\$params['decodeAlias']) { return \$this->alias2id(\$params['value']); } else { return \$this->id2alias(\$params['value']); } } function id2alias(\$value) { return '--'.\$value.'--'; } function alias2id(\$value) { if (ereg('^--[0-9]+--\$', \$value, \$reg)) { return \$reg[1]; } } }</pre> Этот класс меняет id "6" на псевдоним "--6--" при кодировании. При расшифровке, псевдоним "--6--" опять меняется на "6". Конечно в этом мало смысла, но подобная функция может кодировать и расшифровывать любой id/псевдоним!
valueDefault	строка	Значение по умолчанию устанавливается, если сегмент пути не соответствует ни одному значению в "valueMap" или был иначе захвачен для перевода. Примечание: значение по умолчанию применяется ПОСЛЕ любых настроенных "noMatch" обработок (и других, смотрите описание параметра "GETvar")

Пример:

```
1:   'preVars' => array(  
2:     array(  
3:       'GETvar' => 'no_cache',  
4:       'valueMap' => array(  
5:         'no_cache' => 1,  
6:       ),  
7:       'noMatch' => 'bypass',  
8:     ),  
9:     array(  
10:      'GETvar' => 'L',  
11:      'valueMap' => array(  
12:        'dk' => '1',  
13:        'danish' => '1',  
14:        'uk' => '2',  
15:        'english' => '2',  
16:      ),  
17:      'noMatch' => 'bypass',  
18:    ),  
19:  ),
```

Пример показывает настройку с двумя prevars в пути, НО обе они не обязательны (ввиду установки “noMatch” => “bypass”).

Обычно URL на языке по умолчанию подобен этому:

```
123/page.html
```

Затем, если L=1 GETvar установлено, URL будет таким:

```
danish/123/print.html
```

И наконец, если первый сегмент соответствует “no_cache” и “no_cache=1”, переменная GET устанавливается и интерпретация языка GETvar перемещается на сегмент 2:

```
no_cache/danish/123/print.html
```

Понятие обхода несоответствующих значений, дает возможность ошибок при совпадении значений двух соседних конфигураций. Например, ошибки ошибки следуют из обозначения языка “no_cache”, так как это слово зарезервировано при настройке первого сегмента!

Удаление настроек “noMatch” приведет к таким URL:

```
//123/page.html  
/danish/123/page.html  
no_cache/danish/123/print.html
```

Лучшее решение — установка значения по умолчанию для языка:

```
1:   'preVars' => array(  
2:     array(  
3:       'GETvar' => 'no_cache',  
4:       'valueMap' => array(  
5:         'no_cache' => 1,  
6:       ),  
7:       'noMatch' => 'bypass',  
8:     ),  
9:     array(  
10:      'GETvar' => 'L',  
11:      'valueMap' => array(  
12:        'dk' => '1',  
13:        'danish' => '1',  
14:        'uk' => '2',  
15:        'english' => '2',  
16:      ),  
17:      'valueDefault' => 'uk',  
18:    ),  
19:  ),
```

Что приведет к таким результатам:

```
uk/123/page.html  
danish/123/page.html  
no_cache/danish/123/print.html
```

Параметр “no_cache” обходится, но выглядит приятнее.

Пример: “fixedPostVars”

```
1:   'fixedPostVars' => array(
2:     'testPlaceHolder' => array (
3:       array(
4:         'GETvar' => 'tx_extrepmgm_pil[mode]',
5:         'valueMap' => array (
6:           'new' => 1,
7:           'categories' => 2,
8:           'popular' => 3,
9:           'reviewed' => 4,
10:          'state' => 7,
11:          'list' => 5,
12:        )
13:      ),
14:      array(
15:        'condPrevValue' => '2',
16:        'GETvar' => 'tx_extrepmgm_pil[display_cat]',
17:        'valueMap' => array (
18:          'docs' => 10,
19:        ),
20:      ),
21:      array(
22:        'GETvar' => 'tx_extrepmgm_pil[showUid]',
23:        'lookUpTable' => array(
24:          'table' => 'tx_extrep_keytable',
25:          'id_field' => 'uid',
26:          'alias_field' => 'extension key',
27:          'addWhereClause' => ' AND NOT deleted'
28:        )
29:      ),
30:      array(
31:        'GETvar' => 'tx_extrepmgm_pil[cmd]',
32:      )
33:    ),
34:    '1383' => 'testPlaceHolder',
35:  ),
```

Здесь показано, как можно использовать “fixedPostVars”, подобно “preVars”, но после пути к странице. Обычно это используется для одной странице, на которой работает известное расширение. В данном примере in the above example this is the case; страница с id “1383” указывает на конфигурацию “alias” с названием “testPlaceHolder”. Пример разработан для репозитория расширений typo3.org.

Конфигурация устанавливает последовательность 3-4 сегментов виртуального пути. Сначала — главное меню, где число определяет модель отображения строками псевдонимов. Второй сегмент - id отображаемой категории, но обратите внимание, что “condPrevValue” присваивается “2” - что значит, что *только если предыдущее значение было “2”*, то будет интерпретирован этот сегмент, иначе он игнорируется! И наконец, существует uid расширения, настраиваемый для перевода в/из ключей расширения. Это безопасный процесс, так как расширения имеют уникальные ключи. Финальная “команда” определяет уровень меню при отображении отдельных расширений.

Эта конфигурация определяет подобные URL (4 сегмента в разделе “fixedPostVars”):

http://typo3.org/1383/categories/docs/doc_core_cg1/details/

или (только 3 сегмента в разделе “fixedPostVars”, так как первый сегмент без “категорий” / 2)

<http://typo3.org/1383/popular/skin1/details/>

->lookUpTable

Определяет таблицу, используемую для поиска строк перевода id в псевдонимы для GETvars.

Ключ	Тип	Описание
table	строка	Название таблицы
id_field	строка	Название поля в котором хранятся id, обычно целые числа, в том числе "uid"
alias_field	строка	Название поля, содержащего строки соответствующих id псевдонимов. Уникально.
addWhereClause	строка	Дополнительный оператор для запроса поиска. Должен устанавливаться автоматически, даже для "удаленных" полей, так как поиск мог осуществляться перед тем как были доступны любые конфигурации таблиц! Пример значения: "AND NOT deleted"
maxLength	целое	Определяет максимальную приемлимую длину псевдонимов. Если значение псевдонима длиннее этого значения, то будет возвращено оригинальное значение. По умолчанию "100"
useUniqueCache	булева переменная	Если установлено, перевод id в псевдонимы автоматически сохраняется в поисковой таблице, обеспечивающей уникальность; При сохранении просто проверяется, ассоциирован ли псевдоним с другим ID (в той же комбинации таблица/поля) и, если это так, создается уникальный псевдоним, обычно запрашиваемый псевдоним с добавлением числа.
languageGetVar	строка	Установка названия переменной GET, используемой как uid языка (обычно "L"). При установке, ищется заголовок локализации, если таблица поддерживает локализацию с установленными "languageField" и "transOrigPointerField". Для конфигурации НУЖНО дублировать названия полей для "languageField" и "transOrigPointerField" из \$TCA таблицы. Это нужно ввиду того, что во время анализа URL, массив TCA еще не доступен для системы и в нем невозможно выполнить поиск. В этом примере используется таблица современной версии "tt_news". Обратите внимание на четыре выделенных жирным строки, содержащие настройку для генерации url локализованных новостей: <pre>'lookUpTable' => array('table' => 'tt_news', 'id_field' => 'uid', 'alias_field' => 'title', 'maxLength' => 50, 'useUniqueCache' => 1, 'addWhereClause' => ' AND NOT deleted', 'languageGetVar' => 'L', 'languageExceptionUids' => '', 'languageField' => 'sys_language_uid', 'transOrigPointerField' => 'l18n_parent')</pre>
languageField	строка	Тоже, что [ctrl][languageField] в TCA для таблицы просмотра.
transOrigPointerField	строка	Тоже, что [ctrl][transOrigPointerField] в TCA для таблицы просмотра.
languageExceptionUids	строка	Список sys_language uid, если не используется "languageGetVar" (см. ниже).
useUniqueCache_conf['strtolower']	булева переменная	Если установлено, псевдонимы переводятся через strtolower()
useUniqueCache_conf['spaceCharacter']	строка	Обычно, для замены пробелов и т.д. в URL используется символ подчеркивания (_). Здесь Вы можете определить другой символ, например дефис (-).
useUniqueCache_conf['encodeTitle_userProc']	userFunc	Дополнительная обработка псевдонимов перед буферизацией.
enable404forInvalidAlias	булева переменная	Если да, при не найденном соответствии псевдонима и id будет генерироваться 404.
autoUpdate	булева переменная	Если да, (и установлен useUniqueCache) псевдонимы автоматически обновляются.
expireDays	целое	Количество дней, при использовании "autoUpdate", через которое обновляются главные псевдонимы. По умолчанию 60.

Пример

```
'lookUpTable' => array(  
    'table' => 'user_3dsplmxml_bfsbrand',  
    'id_field' => 'xml_id',  
    'alias_field' => 'xml_title',  
    'maxLength' => 10,  
    'addWhereClause' => ' AND NOT deleted'  
)
```

->actionConfig

Ключ	Тип	Описание
type	строка параметров	<p>Действие идентифицируемое по одному из этих ключевых слов:</p> <p>“admin” : возможно редактирование во внешнем интерфейсе (подобно ->postVarSet, type = “admin”).</p> <p>“redirect” : перенаправление по другому URL с параметрами в виде значений сегментов пути и оставшегося пути. Полезно для ссылок на сайте.</p> <p>“notfound” : генерация ошибки 404.</p> <p>“bypass” : обход И добавление ключа снова в стек! (так как ключ принадлежит следующей части URL)</p> <p>“feLogin” : добавляется ключевое слово при входе пользователя. Позволяет использовать отличный URL при регистрации, очень полезно для контроля буферизации заголовков, различая URL в сессии при регистрации пользователей и, соответственно, запрещая серверу буферизацию. Примечание: этот параметр действует при регистрации пользователя; группы, добавляемые по маске IP или иначе не распознаются.</p> <p>По умолчанию: Passthrough(без добавления ключа).</p> <p>Примечание о ключе “_DEFAULT”: если тип “_DEFAULT” не “bypass” (это значит, что действие _DEFAULT что-то <i>делает</i>) метод кодировки URL будет искать первое вхождение действия без типа (переход без добавления ключа в стек) и использует его как значение сегмента.</p>
Only type “redirect”		
url	строка	<p>URL для перенаправления. Существует два маркера, которые может использовать URL:</p> <p>###INDEX### - вставка текущего сегмента.</p> <p>###REMAIN_PATH### - вставка оставшегося пути с текущей точки (включая имя файла)</p> <p>Значение rawurlencoded()</p>

Пример: перенаправление и требуемые префиксы

```

1:   'redirects' => array(
2:     '' => 'cms/',
3:     'mailinglist/' => 'http://lists.netfielders.de',
4:   ),
5:   'preVars' => array (
6:     array(
7:       'type' => 'action',           // "type" действие
8:       'index' => array(
9:         'cms' => '',           // Просто обход
10:        'admin' => array(
11:          'type' => 'admin'     // Переход ко входу в ВЕ ИЛИ установка редактирования во
внешнем интерфейсе...
12:        ),
13:        'search' => array(
14:          'type' => 'redirect',     // Перенаправление...
15:          'url' => 'index.php?id=1344&tx_indexedsearch[sword]=###REMAIN_PATH###',
16:        ),
17:        'ext' => array(
18:          'type' => 'redirect',     // Перенаправление...
19:          'url' => 'cms/1383/list/###REMAIN_PATH###/index.html',
20:        ),
21:        '_DEFAULT' => array(
22:          'type' => 'notfound'     // Если ключ не найден в индексах, генерируется"404".
23:        ),
24:      ),
25:    ),
26:  ),

```

В этом примере первый сегмент URL настроен как действие. Сегмент обязателен, так как “type” индекса “_DEFAULT” установлен “notfound”, т.е., если не найдено соответствующих ключей, генерируется ошибка “Страница не найдена”.

Вот пример, иллюстрирующий подобную конфигурацию, взятый с вебсайта typo3.org:

URL	Что происходит
http://typo3.org/	Такой URL привел бы к ошибке 404, если не было бы настроено перенаправление в строках 1-4. Система перенаправляет на "cms/", если нет виртуального пути.
http://typo3.org/cms/1420/	Будет показана страница с ID 1420. Ключ действия "cms" ничего не делает – просто обходится обработка до следующего уровня, который представляет собой ID страницы. Определяя такой префикс в конфигурации, Вы определяете что сайт "выполняется" из виртуальной директории "cms". Конфигурация в строке 9
http://typo3.org/admin/1420/	Что тоже выведет страницу ID 1420, но активирует редактирование во внешнем интерфейсе, и, если пользователь еще не зарегистрировался как пользователь внутреннего интерфейса, перенаправит на форму входа во внутренний интерфейс. Это настраивается в строках 10-12 При использовании опции редактирования во внешнем интерфейсе ("admin") в "realurl", сегмент пути "admin/" будет присутствовать в ссылках на сайт, таким образом Вы остаетесь в режиме администрирования, пока сами его не удалите. Это также означает, что буферизация страниц отключена и страницы с сегментом "admin/" не сохраняются!
http://typo3.org/search/system+requirements	Перенаправит на http://typo3.org/index.php?id=1344&tx_indexedsearch[sword]=system%2Brequirement , т.е. на страницу с плагином indexed_search и автоматически произведет поиск по слову после сегмента "search". Это настроено в строках 13-16; обратите внимание, что запрашиваемые для поиска слова (оставшийся путь) автоматически вставляются в URL при помощи маркера "###REMAIN_PATH###"
http://typo3.org/ext/lang	Перенаправит на http://typo3.org/cms/1383/list/lang/index.html , где "lang" вставляется маркером "###REMAIN_PATH###", также, как и в действии "search" до того. Принцип тот же. Настройка в строках 17-20
http://typo3.org/maillinglist/	Перенаправит на http://lists.netfielders.de согласно строке 3 конфигурации

Пример: языковой префикс и действие "admin"

```

1:   'preVars' => array (
2:       array(
3:           'type' => 'action',           // "type" действие
4:           'index' => array(
5:               'admin' => array(
6:                   'type' => 'admin' // Переход ко входу в ВЕ ИЛИ редактирование во внешнем интер-
фейсе...
7:               ),
8:               'search' => array(
9:                   'type' => 'redirect', // Перенаправление...
10:                  'url' => 'index.php?id=1344&tx_indexedsearch[sword]=###REMAIN_PATH###',
11:              ),
12:              'ext' => array(
13:                  'type' => 'redirect', // Перенаправление...
14:                  'url' => 'cms/1383/list/###REMAIN_PATH###/index.html',
15:              ),
16:              '_DEFAULT' => array(
17:                  'type' => 'bypass' // Если ключ не найден в индексах, осуществляется переход.
18:              ),
19:          ),
20:      ),
21:      array(
22:          'GETvar' => 'L',
23:          'valueMap' => array(
24:              'dk' => '1',
25:          ),
26:          'noMatch' => 'bypass',
27:      ),
28:  ),

```

Здесь настроены две preVars, первая содержит действия, почти как в предыдущем примере, за исключением настройки "_DEFAULT" типа "bypass", означающей, что, если ключ не найден, произойдет переход к расшифровке следующей preVar настройке для этого сегмента (обход и добавление значения сегмента снова в стек).

Кроме того, настроен языковой префикс.

Результат такой конфигурации должен быть ясен из следующей таблицы:

URL	Что происходит
http://typo3.org/1420/	Показана страница 1420
http://typo3.org/admin/1420/	Показана страница 1420 с возможностью редактирования во внешнем интерфейсе
http://typo3.org/dk/1420/	Показана страница 1420 на датском (&L=1)
http://typo3.org/admin/dk/1420/	Показана страница 1420 на датском (&L=1) с возможностью редактирования во внешнем интерфейсе

Пример: передача feLogin

Здесь дается префикс “loginarea/” для всех URL при регистрации пользователя внешнего интерфейса. Сам по себе префикс ничего не делает; никаких внутренних параметров. Но он крайне важен для управления буферизацией, так как позволяет отправлять буферизованные заголовки всех страниц клиентскому браузеру, если пользователь не зарегистрирован, а страницы зарегистрированного пользователя имеют другой URL (основной URL с добавленным ключевым словом), для них можно запретить буферизацию.

Пример настройки:

```
'preVars' => array(
    array(
        'type' => 'action',
        'index' => array(
            'loginarea' => array(
                'type' => 'feLogin'
            ),
            '_DEFAULT' => array(
                'type' => 'bypass'
            )
        )
    ),
),
```

->pagePath

Настройка метода кодирования/расшифровки id в/из “пути к странице”

Ключ	Тип	Описание
type	строка	Настройка метода кодирования/расшифровки ID. По умолчанию — установка просто id/псевдонима страницы, как элемент виртуального пути. “user” : вызывает внешний класс для генерации.
Только для типа “user”:		
userFunc	ссылка на функцию	Сылка на функцию обработки кодирования id. Примеры можно найти в class.tx_realurl_dummy.php Полная реализация в “class.tx_realurl_advanced.php”. Детальнее об этом говорится позже в этой документации. Пример значения: EXT:realurl/class.tx_realurl_advanced.php:&tx_realurl_advanced->main
rootpage_id	целое	Определяет uid корневой страницы настраиваемого сайта. Это принудительный параметр при работе нескольких сайтов на одной базе данных при использовании понятных url для нескольких сайтов. Этот параметр делает возможным отличать сайты по алгоритму “путь-в-id”. Например, у двух сайтов может быть одна главная страница на первом уровне, поэтому будет сформирован тот же понятный путь url. При этом, два разных ID связаны с одним путем и нужно их как-то отличить. Для настройки разных сайтов, просто сделайте другую конфигурацию для разных доменов. Смотрите раздел об основной настройке. Очень важно использовать уникальный id корневой страницы сайта. Иначе не будет работать просмотр страниц из восстановленных путей. Эти установки не нужны, если realurl настраивается для одного сайта. Смотрите пример в разделе tx_realurl_advanced.
[Другие ключи зависят от пользовательских функций]		

Пример перевода id в путь:

```
'pagePath' => array(
    'type' => 'user',
    'userFunc' => 'EXT:realurl/class.tx_realurl_advanced.php:&tx_realurl_advanced->main',
    'spaceCharacter' => '-',
    'languageGetVar' => 'L',
    'expireDays' => 30
),
```

Вызывается пользовательская функция, формирующая понятный URL по заголовку страницы, а не просто вывод ее id номер. Подробное описание этого класса приводится ниже в этой документации, в разделе “class.tx_realurl_advanced.php”

URL, сформированный вышеприведенной настройкой выглядит подобно этому (перед / после):

```
1420/index.html
extensions/index.html

1420/repository/popular/skin1/details/index.html
extensions/repository/popular/skin1/details/index.html

1440/index.html
documentation/glossary/index.html

1409/index.html
about/license/gpl-for-developers/index.html

1342/showreference/52/
about/yet-another-typo3-site/showreference/52/
```

Пример фиктивной настройки:

```
'pagePath' => array(
    'type' => 'user',
    'userFunc' => 'EXT:realurl/class.tx_realurl_advanced.php:&tx_realurl_dummy->main',
),
```

Вызывает класс dummy, повторяющий действия основного: выводит id/псевдонимы страниц и ничего более. Но на его основе можно реализовать любые нужные Вам схемы!

->postVarSet

Ключ	Тип	Описание
type	строка параметров	По умолчанию postVarSet включает <ul style="list-style-type: none">● ключевое слово, идентифицирующее следующий раздел виртуального пути● соответствие GETvars одному или нескольким сегментам пути Пример (из предыдущих): "news/list/456/", где ключ "news/" (первый сегмент) идентифицирует следующие разделы ("list/456/") в соответствующие GET-vars tx_mininews[mode] и tx_mininews[showUid] соответственно (согласно настройкам). Настройка последовательности разделов сделана в числовом массиве ->partDef. Другие режимы: Существуют другие режимы, включаемые установкой следующих ключевых слов для параметра "type". При этом, после ключевого слова не может быть никакой последовательности разделов, но будет вызван альтернативный режим. "single" : используя этот ключ, Вы связываете ключ с точным количеством GETvar с конкретными значениями. Это работает подобно привязыванию имен файлов к GETvar (смотрите ->fileName) "admin" : используя этот ключ, Вы сделаете доступной для пользователя редактирование сайта во внешнем интерфейсе, показав контекстно зависимые иконки редактирования. Если пользователь не зарегистрировался во внутреннем административном интерфейсе, он будет перенаправлен на страницу входа в систему и, после успешной регистрации, возвращен обратно на ту же страницу.
Только для типа по умолчанию:		
[0..x]	->partDef	Настройка раздела, связанного с "ключом" определяемым этим postVarSet.
Только для типа "single":		
keyValues	array of [GETvar] => [string value]	Массив "keyValues" определяет одну или несколько переменных GET с <i>определенными</i> значениями. Ключ postVarSet соответствует, если <i>все</i> GET-vars, настроенные в ["keyValues"] находятся в оставшихся переменных GET, нуждающихся в переводе, и значения точно соответствуют!

Пример: редактирование во внешнем административном интерфейсе

```
1:     'postVarSets' => array (
2:         '_DEFAULT' => array(
3:             ....
4:             'edit_now' => array(
5:                 'type' => 'admin'
6:             )
7:         ),
8:     )
```

Добавление строк 4-6 в приведенный выше фрагмент кода настройки postVarSets, сделает доступным редактирование во внешнем интерфейсе, при добавлении ".../edit_now" к виртуальному пути. Конечно, можно выбрать любую "admin-directory".

Одно предостережение; если ключ будет добавлен к URL, когда не закончен предыдущий раздел postVarSet, то ключ, конечно же, будет выглядеть как параметра этой postVarSet, а не как ключ для перехода к желаемой модели редактирования во внешнем интерфейсе. Ввиду этого, Вы возможно захотите использовать эту возможность, но для pre-vars.

Пример: PostVarSets

```
1:     'postVarSets' => array(  
2:         '_DEFAULT' => array(  
3:             'plaintext' => array(  
4:                 'type' => 'single', // Особый режим postVars  
5:                 'keyValues' => array(  
6:                     'type' => 99  
7:                 )  
8:             ),  
9:             'ext' => array(  
10:                array(  
11:                    'GETvar' => 'tx_myExt[p1]',  
12:                ),  
13:                array(  
14:                    'GETvar' => 'tx_myExt[p2]',  
15:                ),  
16:                array(  
17:                    'GETvar' => 'tx_myExt[p3]',  
18:                ),  
19:            ),  
20:             'news' => array(  
21:                array(  
22:                    'GETvar' => 'tx_mininews[mode]',  
23:                    'valueMap' => array(  
24:                        'list' => 1,  
25:                        'details' => 2,  
26:                    )  
27:                ),  
28:                array(  
29:                    'GETvar' => 'tx_mininews[showUId]',  
30:                ),  
31:            ),  
32:         ),  
33:     ),
```

Здесь показано, как настроены три набора `postVarSets`, два из них имеют тип по умолчанию (ключ + раздел `GETvars`), в то время как третье имеет тип `“single”`, т.е. соответствие фиксированному значению `GETvar`.

Для разъяснения этой настройки и эффекта от нее, нужно изучить следующие примеры с пояснениями на основе вышеприведенной конфигурации. В каждом — две строки, URL с параметрами `GET` и его понятная версия.

```
index.php?id=123&tx_myExt[p3]=ccc&tx_myExt[p2]=bbb&tx_myExt[p1]=aaa  
123/ext/aaa/bbb/ccc/
```

Для кодирования параметров `GET` выше, использовалась `postVarSet “ext”`. Раздел иницируется ключем `“ext”` и следующие три сегмента виртуального пути соответствуют трем переменным `GET`, настроенным для этого ключевого слова (строки 10-18).

```
index.php?id=123&tx_myExt[p1]=aaa  
123/ext/aaa/
```

```
index.php?id=123&tx_myExt[p1]=aaa&tx_myExt[p2]=bbb  
123/ext/aaa/bbb/
```

Эта пара примеров показывает, что при использовании лишь одного или двух параметров, пустые просто удаляются с конца пути. Первый пример должен выглядеть как `“123/ext/aaa/”`, а второй, соответственно, `“123/ext/aaa/bbb/”`, но ввиду того, что пустое значение находится в конце, мы можем его безпрепятственно удалить, как и видно из примера.

```
index.php?id=123&tx_myExt[p1]=aaa&tx_myExt[p3]=ccc  
123/ext/aaa//ccc/
```

Здесь использованы только `“tx_myExt[p1]”` и `“tx_myExt[p3]”`, а используемый в понятном пути промежуточный сегмент `“p2”` отсутствует, вместо него мы получаем пустой сегмент виртуального пути.

```
index.php?id=123&tx_mininews[showUId]=123&tx_mininews[mode]=1  
123/news/list/123/
```

В этом примере, параметр `mininews` кодируется словом `“news”`. Помните, что `“tx_mininews[mode]”` `GETvar` имеет таблицу отображения, которая позволяет автоматически переводить значение `“1”` в используемое в виртуальном пути `“list”`. Эта (и подобные) особенности позволяют создавать истинно понятные URL даже для номеров ID.

```
index.php?  
id=123&tx_mininews[showUId]=123&tx_mininews[mode]=1&tx_myExt[p1]=aaa&tx_myExt[p2]=bbb&tx_myExt[p3]=ccc  
123/ext/aaa/bbb/ccc/news/list/123/
```

В этом примере две `postVarSets`, `“ext”` и `“news”`. Как видно, длина последовательности сегментов не вызывает проблем и следующий сегмент успешно регистрируется.

Помните, что ключ `“ext”` становится начальным. Это происходит из-за того, что настройка для `“ext”` `postVarSet` находится в начале и обрабатывается перед `“news”` `postVarSet`.

```
index.php?id=123&tx_mininews[showUid]=123&tx_myExt[p1]=aaa&tx_myExt[p3]=ccc
123/ext/aaa//ccc/news//123/
```

Этот пример похож на предыдущий, за исключением двух отсутствующих параметров, ввиду чего два пустых сегмента присутствуют в виртуальном пути. Эту проблему разрешить не удастся, так как должна соблюдаться длина последовательности, например, ввиду того, что "mode" для "news" postVarSet была определена перед параметром "showUid", удалить пустые значения невозможно.

```
index.php?id=123&type=99&tx_myExt[p1]=aaa&unknownGetVar=foo
123/plaintext/ext/aaa/?unknownGetVar=foo
```

Этот пример немного отличается от предыдущих, показывая, что случается, когда присутствует неизвестная переменная GET. Она просто добавляется в конец URL, и все!

->fileName

Настройка значений имен файлов в виртуальном пути.

Ключ	Тип	Описание
index[filename]["keyValues"]	массив [GETvar] => [string value]	"index" - массив виртуальных названий файлов (напр. "page.html"), соответствующих одной или нескольким переменным GET с <i>определенными</i> значениями. Имя файла выбирается при кодировании, если <i>все</i> настроенные в index[filename] ["keyValues"] переменные GET находятся в оставшихся для перевода переменных GET и значения точно соответствуют! При отсутствии совпадений, используется значение "_DEFAULT". Важно: очень важен порядок имен файлов, так как будет выбрано первое попавшееся соответствие. Ввиду чего нужно хорошо его продумать. Смотрите пример ниже.
defaultToHTMLsuffixOnPrev	булева переменная/строка	Если установлено, последняя часть виртуального пути будет превращена в имя файла, добавлением суффикса ".html", ЕСЛИ такого файла не существует. Например, "workplace-learning-solutions/companion-solutions/" превратится в "workplace-learning-solutions/companion-solutions.html" и основная часть имени файла (исключая расширение ".html") будет восприниматься как последняя часть виртуального пути. Это полезно для симуляции HTML документов, даже без настройки отображения имен файлов. Если установлена строка, она будет использована как суффикс. Обратите внимание, что обязательно должна присутствовать точка (т.е. правильно ".html", а не "html")
acceptHTMLsuffix	булева переменная/строка	Если URL содержит суффикс ".html" и этот параметр включен, суффикс будет удален из url. Так, Вы сможете легко перейти со старого сайта на ТУР3, оставив все внешние ссылки рабочими, используя url с другим суффиксом (через defaultToHTMLsuffixOnPrev) или вообще без суффикса.

Пример: различные имена файлов для frameset

```
1:   'fileName' => array (
2:       'index' => array(
3:           'print.html' => array(
4:               'keyValues' => array (
5:                   'print' => 1,
6:                   'type' => 1,
7:               )
8:           ),
9:       'page.html' => array(
10:          'keyValues' => array (
11:              'type' => 1,
12:          )
13:      ),
14:      'top.html' => array(
15:          'keyValues' => array (
16:              'type' => 2,
17:          )
18:      ),
19:      '_DEFAULT' => array(
20:          'keyValues' => array(
21:              )
22:          ),
23:      ),
23:      'acceptHTMLsuffix' => '.cfm'
24:  ),
```

Этот пример можно использовать как конфигурацию веб-сайта на основе фреймов. Когда не значение "type" не нужно, используется ключ по умолчанию (который конечно не представляет имени файла), но, если значение &type, 1 или 2, используются либо "page.html", либо "top.html"; они будут представлять переменную GET "&type=1" и "&type=2" соответственно во время кодирования.

Обратите внимание, как параметр "&print=1" переводится в имя файла! Идея в том, что "print.html" используется, если есть две переменные GET; при этом "&type=1" и "&print=1". *Осторожно, упорядочивайте* имена файлов; если бы "print.html" находился бы ниже "page.html", то он никогда бы не использовался, так как используется первое найденное значение, и это было бы "page.html", так как "&type=1", а переменная GET "&print=1" была бы добавлена к URL (например "page.html?print=1"), вместо того, чтобы стать названием файла ("print.html").

Пример: имя файла по умолчанию

```
'fileName' => array (
  'index' => array(
    'index.html' => array(
      'keyValues' => array(
        )
      )
    )
  )
),
```

В этой конфигурации, файл “index.html” будет префиксом независимо ни от чего.

Обработка относительных ссылок при понятных URL

По умолчанию, ссылки на другую страницу в TYPO3 формируются так www.server.com/index.php?id=123&type=0, как будто бы все станицы расположены в одной (файловой системе-) директории: корне сайта. Проблема в том, что многие расширения (в том числе ядро TYPO3) работают с изображениями, javascripts, и т.п., определяя путь к ним относительно корня TYPO3, например “typo3/ext/indexed_search/pi/res/pages.gif”. Этот подход не работает, если путь постоянно меняется.

Например, файл “fileadmin/my_image.jpg” относительно “index.php?id=123” будет найден, так как “index.php” находится в корневой папке, где и расположена папка “fileadmin/”. Но, как только URL “index.php?id=123” переводится в понятный URL, “contact/company_address/”, Ваш браузер попытается найти картинку в “contact/company_address/fileadmin/my_image.jpg” и конечно же не найдет.

Для разрешения этой проблемы Вы

1. либо делаете префикс с абсолютным путем к корню сайта для всех относительных ссылок
2. или настраиваете тег <base> в заголовке HTML файлов к корню сайта.

config.absRefPrefix

Существует директива в настройке TypoScript, для установки абсолютного префикса для всех ссылок или изображений (config.absRefPrefix), но она не везде реализуется (например индексированный поиск или редактирование во внешнем интерфейсе), и, таким образом плохо работает.

Не используйте config.absRefPrefix. У него есть плохие свойства, приводящие к неработоспособности RealURL. Одна из проблем — страница 404 TYPO3 без тега <base>, на которой не будет логотипа TYPO3:)

Можно использовать этот метод, исправив все ошибки, но это требует, чтобы все относительные ссылки формировались однообразно, что конечно не может быть гарантировано для всех расширений.

Тег <base>

Простое решение через HTML: нужно лишь прописать тег <base> в <head> Ваших страниц, например:

```
<base href="http://your.domain.com/">
```

Чтобы сделать Ваши шаблоны TypoScript приемлимыми для RealURL, нужно включить этот оператор в Ваш шаблон HTML или использовать следующий код TypoScript:

```
config.baseURL = http://your.domain.com/
```

Это автоматически прочтется, как базовый URL Вашего сайта (используя t3lib_div::getIndpEnv('TYPO3_SITE_URL')) и <base> тег будет создан в заголовке HTML при выводе во внешний интерфейс.

Метод <base> тега, по всей видимости безупречно работает в TYPO3, кроме двух случаев, когда есть такая ссылка - она не будет работать, потому что она будет ссылаться на корень сайта, а не перенаправлять внутри документа.

Можно просто решить эту проблему:

```
config.prefixLocalAnchors = all
```

Нужный префикс будет установлен для все экземпляров '<a href="#...."....' страницы; в основном везде, где генерируются локальные привязки. Эта подстановка производится ерег_replace в общем контенте страницы после отображения. За деталями обратитесь к TSref.

Другая ситуация специфична для MSIE; при установке "document.location" через JavaScript, MSIE понимает относительные ссылки URL, как относительные к текущему URL, а не к <base> URL. Поэтому у базового URL появится префикс. Вы можете найти это значение в \$GLOBALS['TSFE']->baseUrl (или использовать t3lib_div::getIndpEnv("TYPO3_SITE_URL")).

Предупреждение: опасно устанавливать “config.baseURL = 1” (автоматическое создание базового URL), если Вы можете обращаться к сайту с IP адреса или внутренней сети. В этом случае базовый URL может быть “http://192.168.1.123/my_site” что а) не будет работать для любых внешних посетителей сайта и б) показывает информацию о структуре директорий сервера (секретность). Эта страница не появится, если страницы не буферизируются посещениями из внутренней сети, но, *если* страницы помещаются в буфер во время посещения сайта из внутренней сети, внутренний базовый URL будет присутствовать на страницах и будет виден внешним посетителям сайта,

отсюда - проблема! То же самое происходит, если страница генерируется и помещается в буфер, как "index.php?id=123", а затем запрашивается по понятному URL, страница, сформированная из "неправильного" местоположения будет содержать неправильное содержимое из буфера.

Создание расширений, совместимых с "config.baseURL"

Если Вы установили "config.baseURL" и затем "config.prefixLocalAnchors = all", расширения могли сформировать неправильные локальные привязки. Это происходит из-за того, что расширения включают не кешируемое содержимое страницы через USER_INT или USER_EXT сObjects, которые не обрабатывают содержимое! (если "config.prefixLocalAnchors" не установлен на "output"). Для таких расширений должна быть сделана поддержка для "realurl", что может быть сделано (с полной обратной совместимостью) добавлением префикса ко всем локальным привязкам, как результат этого:

```
substr(t3lib_div::getIndpEnv('TYPO3_REQUEST_URL'), strlen(t3lib_div::getIndpEnv('TYPO3_SITE_URL')));
```

или в последних версиях TYPO3:

```
$GLOBALS['TSFE']->anchorPrefix
```

В общем

Удостоверьтесь, что настройки работают для ВСЕХ формируемых типов страниц!

class.tx_realurl_advanced.php

Введение

Класс tx_realurl_advanced предоставляет дополнительную кодировку ID страниц в пути, включая кодирование в локализованные заголовки и управление буферизацией.

Настройка

Нужно создать записи домена на начальных страницах домена. Даже если домен один, для него нужно создать запись. При этом нужно помнить одну вещь:

Если TYPO3 установлен в корневую директорию для документов хоста, нужно создать запись домена по типу 'www.server.com'. Если же TYPO3 установлен в другую директорию, запись для домена должна быть вроде этой 'www.server.com/the_path_to_your_typo'. Слеш в конце не обязателен.

Настройка "realurl" для работы с "tx_realurl_advanced" ID кодировкой

Просто установите эту настройку для ключа "pagePath" в массиве настроек:

```
'pagePath' => array(
    'type' => 'user',
    'userFunc' => 'EXT:realurl/class.tx_realurl_advanced.php:&tx_realurl_advanced->main',
    'spaceCharacter' => '-',
    'languageGetVar' => 'L',
    'expireDays' => 30
),
```

Вот директивы, специфичные для "tx_realurl_advanced":

Директива:	Тип данных:	Описание:
languageGetVar	строка	Определяет переменную GET в URL, отвечающую за id языка; если установлено, значение языка будет принято во внимание и будет предпринята попытка создания локализованной версии пути.
languageExceptionUids	строка	Список UID записей sys_language, для которых не формируется локализованный URL при установленной languageGetVar.
spaceCharacter		Обычно, символ подчеркивания (_) используется для замены пробелов и подобных символов в URL. Здесь Вы можете определить другой символ, например дерфис (-).
segTitleFieldList	список названий полей	Расположенный в порядке приоритета список полей таблицы страниц (<i>корневых!</i>), используемых для формирования понятного URL. По умолчанию "tx_realurl_pathsegment,alias,nav_title,title" (для записей альтернативного языка страниц только "nav_title, title"). Примечание: если Вы определили свои поля, которых нет в корне, нужно добавить их туда через "\$GLOBALS['TYPO3_CONF_VARS']['FE']['addRootLineFields']"
disablePathCache	булева переменная	Отменяет буферизацию страниц. Система будет полагаться исключительно на предварительный поиск в полях "segTitleFieldList".
autoUpdatePathCache	булева переменная	(Зависит от сброшенного значения "disablePathCache!") Если установлено, URL автоматически обновляются после изменения заголовка страницы, псевдонима и других, определенных в "segTitleFieldList" полях и старые значения будут прослеживаться в течение времени, определенного в "expireDays" (смотрите ниже). Расплатой за это служит то, что pathCache не используется для ускорения поиска, а, вместо этого, отслеживаются старые URL. В угоду производительности, "disablePathCache" отключен.
expireDays	целое	Время (в днях), в течение которого будут прослеживаться старые URL страниц с измененными заголовками и т.д. Смотрите "autoUpdatePathCache". По умолчанию 60.
firstHitPathCache	булева переменная	Если установлено, поиск пути в буфере производится, только при первом результирующем результате. В версиях до 1.4 это предотвращало ошибку "was not set as postVarSet as expected" ценой дополнительного поиска в базе данных. В версиях после 1.4 больше не нужно включать этот параметр, оставленный ввиду совместимости.
rootpage_id	целое	Этот параметр важен для должного функционирования tx_realurl_advanced в многодоменной среде. Описание можно найти в разделе ->pagePath . В версиях до 1.0.1 это определялось в tx_realurl_advanced, но сейчас производится через настройку realurl. Смотрите пример ниже.
encodeTitle_userProc	функция пользователя	Дополнительная пользовательская обработка в "encodeTitle()".
dontResolveShortcuts	булева переменная	Если установлено, страницы-ссылки не обращаются к целевым страницам. NB: Если не установить этот параметр, модуль управления понятными URL внутреннего интерфейса может сообщить о дублировании в pathCache таблице, ввиду того, что и целевая, и страница ссылка имеют тот же URL. Конечно это не реальная ошибка, но режет глаз.
excludePagelds	строка	Список разделенных запятой (БЕЗ пробелов!) id страниц, не обрабатываемых realurl.

Пример: настройка realurl для нескольких доменов в одной базе данных

```

1: $TYPO3_CONF_VARS['EXTCONF']['realurl'] = array(
2:     '_DEFAULT' => array(
3:         'pagePath' => array(
4:             'type' => 'user',
5:             'userFunc' => 'EXT:realurl/class.tx_realurl_advanced.php:&tx_realurl_advanced->main',
6:             'rootpage_id' => 437
7:         ),
8:     ),
9:     'www.test1.intra' => array(
10:        'pagePath' => array(
11:            'type' => 'user',
12:            'userFunc' => 'EXT:realurl/class.tx_realurl_advanced.php:&tx_realurl_advanced->main',
13:            'rootpage_id' => 111
14:        ),
15:    )
16: );

```

Обратите внимание, что поля rootpage_id различаются для двух случаев!

Автоматическая настройка

Обзор

Начиная с версии 1.4, RealURL имеет параметр автонастройки. Которая, после проверки системы, попытается создать файл настроек для упрощения работы пользователя. После проверки пользователю не нужно делать ручную настройку. RealURL старается сделать оптимальную настройку для наиболее общих случаев. Если для Вашего хоста требуются определенные настройки, Вы должны сделать их вручную или изменить сформированные автоматически.

RealURL создает настройки для каждого домена и языка, определенного в системе. Для создания языковых настроек требуется расширение `static_tables`. `static_tables` предлагается установить при установке RealURL. Если Вам не нужна автонастройка или используется только один язык, можно отказаться от `static_tables`.

RealURL может сохранить сформированную конфигурацию в одном из двух форматов: сериализованный массив PHP или PHP код. Эти параметры предлагаются при установке расширения. В первом случае все работает примерно в 10 раз быстрее (или меньше при использовании PHP акселератора) и должно использоваться в производственной среде. Но сериализованный массив не может быть (ни при каких обстоятельствах!) изменен вручную. Он может перестать работать после модернизации версии PHP, хотя это редкий случай. В любом случае, пользуйтесь вторым параметром только если:

- Вы планируете изменять созданную настройку
- Вы планируете обновить версию PHP в ближайшем будущем

Важно: настройка создается однажды. При добавлении домена или языка в систему, автоконфигурация не обновляется. Вы должны удалить файл `realurl_autoconf.php` в директории `typo3conf/`. RealURL обновит конфигурацию, если этого файл не существует.

Если Вы используете ручную настройку, автонастройку можно отключить сняв флажок при установке расширения. Это сэкономит несколько миллисекунд.

Автоматическая настройка расширения

Расширения могут изменить автонастройку RealURL под свои потребности. Помните, что это делается однажды, при формировании настроек. Если вы устанавливаете расширение, поддерживающее автогенерацию, нужно удалить файл `realurl_autoconf.php` в директории `typo3conf/`. RealURL обновит конфигурацию, если этого файл не существует.

Для добавления/изменения настроек RealURL, расширения должны предоставить обработчик для RealURL. Этот обработчик вызывается RealURL для формирования массива. Массив — это *шаблон*, а не конечная настройка. RealURL использует этот массив для формирования различных наборов массивов для каждого домена. Воспринимайте этот массив, как запись `_DEFAULT` в настройке RealURL.

Следующий пример, взятый из расширения `album3x`, показывает как установить такой обработчик в `locaconf.php`:

```
// RealURL autoconfiguration
$GLOBALS['TYPO3_CONF_VARS']['SC_OPTIONS']['ext/realurl/class.tx_realurl_autoconfgen.php']
['extensionConfiguration']['album3x'] = 'EXT:album3x/class.tx_album3x_realurl.php:tx_album3x_realurl->addAlbum3xConfig'
```

Вот пример обработчика из расширения `album3x`:

```
class tx_album3x_realurl {

    /**
     * Generates additional RealURL configuration and merges it with provided configuration
     *
     * @param array $params Default configuration
     * @param tx_realurl_autoconfgen $pObj Parent object
     * @return array Updated configuration
     */
    function addAlbum3xConfig($params, &$pObj) {
        return array_merge_recursive($params['config'], array(
            'postVarSets' => array(
                '_DEFAULT' => array(
                    'page3x' => array(
                        array(
                            'GETvar' =>
                                'tx_album3x_pil[page]',
                        ),
                    ),
                    'image3x' => array(
                        array(
                            'GETvar' =>
                                'tx_album3x_pil[showUid]',
                            'userFunc' =>
                                'EXT:album3x/class.tx_album3x_realurl.php:&tx_album3x_realurl->main',
                        ),
                    ),
                ),
            ),
        ));
    }
}
```

```
}
```

Важное замечание: попробуйте придумать имена для `preVar`, `postVar` и `fixedPostVar`, которые не конфликтовали бы с другими расширениями. В тоже время, названия переменных должны хорошо выглядеть в URL, иначе могут произойти конфликты. Например, расширение `mininews` пытается предотвратить конфликт с `tt_news` изменяя название `postVar`, если установлены `tt_news`:

```
class tx_mininews_realurl {  
  
    /**  
     * Generates additional RealURL configuration and merges it with provided configuration  
     *  
     * @param array $params      Default configuration  
     * @param tx_realurl_autoconfgen $pObj      Parent object  
     */  
    function addMininewsConfig($params, &$pObj) {  
        $postVar = (t3lib_extMgm::isLoaded('tt_news') ? 'mnews' : 'news');  
        return array_merge_recursive($params['config'], array(  
            'postVarSets' => array(  
                '_DEFAULT' => array(  
                    $postVar => array(  
                        'GETvar' =>  
  
                    'tx_mininews_pil[showUid]'  
  
                )  
            )  
        ));  
    }  
}
```

Приложение

mod_rewrite замечания от Ole Tange, www.fi.dk

Используя `RealUrl`, нужно переписать правила в `.htaccess` (или `apache.conf`) для переназначения пути к `index.php`. Это можно сделать так:

```
RewriteRule .* /index.php
```

Однако, `Apache` не должен перезаписывать документы, расположенные в `/typo3/`, `/uploads/`, `/fileadmin/` и `/typo3conf/`:

```
RewriteRule ^typo3/.*$ - [L]  
RewriteRule ^uploads/.*$ - [L]  
RewriteRule ^fileadmin/.*$ - [L]  
RewriteRule ^typo3conf/.*$ - [L]
```

Как и в `/typo3` (без `'/'` в конце):

```
RewriteRule ^typo3$ - [L]
```

Если Вы хотите дать хороший URL для PDF файла (например `www.example.com/project-name/report.pdf`) вы захотите вставить его в `/project-name/`. Существующие файлы также не должны быть переписаны:

```
RewriteCond %{REQUEST_FILENAME} !-f
```

Используемые симлинки к фалу также не должны быть переписаны:

```
RewriteCond %{REQUEST_FILENAME} !-l
```

Если `www.example.com/project-name` должен рассматриваться как каталог, чтобы относительные ссылки на `'background'` были бы ссылками на www.example.com/project-name/background, а не на `www.example.com/background`

Просто добавьте `'/'` к url, если это не файл:

```
RewriteRule (.*[/])$ http://%{HTTP_HOST}/$1/ [L,R]
```

Это работает как переадресация с `www.example.com/project-name` на `www.example.com/project-name/`

Если в директории `project-name` есть `index.html`, нужно использовать его, вместо `Typo3`:

```
RewriteCond %{REQUEST_FILENAME}/index.html -f  
RewriteRule / %{REQUEST_URI}/index.html [L]
```

То же самое для `index.htm` и `index.php`:

```
RewriteCond %{REQUEST_FILENAME}/index.htm -f  
RewriteRule / %{REQUEST_URI}/index.htm [L]
```

```
RewriteCond %{REQUEST_FILENAME}/index.php -f
```

```
RewriteRule / %{REQUEST_URI}/index.php [L]
```

В результате .htaccess выглядит примерно так:

```
RewriteEngine On
RewriteRule ^typo3$ - [L]
RewriteRule ^typo3/.*$ - [L]
RewriteRule ^uploads/.*$ - [L]
RewriteRule ^fileadmin/.*$ - [L]
RewriteRule ^typo3conf/.*$ - [L]

RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-l
RewriteRule (.*/[^/])$ http://%{HTTP_HOST}/$1/ [L,R]

RewriteCond %{REQUEST_FILENAME}/index.html -f
RewriteRule / %{REQUEST_URI}/index.html [L]

RewriteCond %{REQUEST_FILENAME}/index.htm -f
RewriteRule / %{REQUEST_URI}/index.htm [L]

RewriteCond %{REQUEST_FILENAME}/index.php -f
RewriteRule / %{REQUEST_URI}/index.php [L]

RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-l
RewriteRule .* /index.php
```

Помните, что здесь важен порядок. Изменение порядка может привести к непредсказуемым результатам.

Использование httpd.conf для настройки, совет от Stefan Bühler (взято из bugtracker)

Есть решение для httpd.conf:

```
-----
RewriteRule ^/(typo3|typo3temp|typo3conf|t3lib|tslib|fileadmin|uploads|showpic\.php) (/.*)?$ - [L]
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-l
RewriteRule ^/(.*)$ /index.php [L,E=ORIG_SCRIPT_NAME:/index.php]
-----
```

Что устанавливает переменную окружения "ORIG_SCRIPT_NAME" на "/index.php", используемую Typo3 (если она существует) вместо "SCRIPT_NAME".

"SCRIPT_NAME" изменяется на нее, при использовании .htaccess с RewriteBase, и невозможно (я знаю и пробовал) изменить из httpd.conf.

Если Typo3 доступен, например через http://host/path-to-typo3/, [^] нужно изменить RewriteRule на:

```
RewriteRule ^/(.*)$ /index.php [L,E=ORIG_SCRIPT_NAME:/path-to-typo3/index.php]
```

Список ToDo

- Технический список todo смотрите в "doc/TODO.txt" папки расширения

- Можно сделать интеллектуальные 404 страницы, используя встроенный индексированный поиск, например. При запросе языка (/nl/...), можно вывести страницу на этом языке. Что-то вроде этого.

Список изменений

- 0.0.1: Первое обновление
- 0.0.4: Первая рабочая версия
- 0.0.5: Добавлена иконка, спасибо Netcreators!
- 0.0.6: Добавлена документация в формате StarOffice
- 0.0.7: Почти полная переработка / пересмотр кода, реализации '/path_to_typo'-возможность, реализация поддержки многодоменной структуры, изменение кода для почти полной автоматизации настроек, обновление документации
- 0.1.0: Первая публично доступная версия
- 0.1.1: pathPrefix не работает должным образом, был добавлен хак для возможности работы
- 0.1.2: URLs in pages weren't rendered correctly on single-language-sites
- 0.1.3: Возможно выбрать символ (вместо подчеркивания) для замены пробела, исправлены другие мелкие ошибки при формировании некоторых URL
- ОБЩАЯ переработка Kasper Skårhøj в марте 2004
- Смотрите список изменений внутри расширения (стандартный список в CVS)