

RealURL

Extension Key: **realurl**

Copyright 2003-2008

Dmitry Dulepov <dmitry@typo3.org>

This document is published under the Open Content License
available from <http://www.opencontent.org/opl.shtml>

The content of this document is related to TYPO3
- a GNU/GPL CMS/Framework available from www.typo3.com

Table of Contents

RealURL	1	class.tx_realurl_advanced.php	22
Introduction	1	Introduction.....	22
What does it do?.....	1	Configuration.....	22
Features.....	2	Hooks.....	23
Configuration	2	Automatic configuration	24
Installation.....	2	Overview.....	24
URL en/decoding background.....	3	Automatic configuration of extensions.....	24
Configuration directives.....	5	Appendix	25
Handling relative links with Speaking URLs.....	20	ToDo list	27
		Changelog	27

Introduction

What does it do?

The extension provides automatic transformation of URLs with GET parameter in the frontend (like "index.php?id=123&type=0&L=1") into a virtual path, a so called "Speaking URL" (like "dutch/contact/company-info/page.html") and back again. The objective is that URLs shall be as human readable as possible.

The extension is very flexible and can provide from simple translation of page IDs to encoding of almost any possible combination of GET parameters.

Examples

Typed URL	TYPO3 id and type
http://www.domain.com/	id=0, type=0
http://www.domain.com/products/product1/features/	id=123, type=0
http://www.domain.com/products/product1/features/leftframe.html	id=123, type=2

Background

TYPO3 works with page-IDs. This works great, however the URLs are very ugly ("...index.php?id=123&type=0&L=1...." etc.). There are workarounds (simulateStaticDocuments), but that's just a fake: the ID must still be supplied in the URL, which is not desirable. Furthermore, only the page-title is shown, not the complete 'path' (or 'rootline') to the page.

Normally, you type in the path and filename of a document, but TYPO3 works exclusively with page-IDs. The RealURL-extension provides a way to translate between page-IDs and (virtual) URLs that are easy to read and remember.

The extension requires the Apache module "mod_rewrite" to rewrite the virtual URLs of the site to the TYPO3 frontend engine. Generally it will work out-of-the-box but you will have to address the issue that all media referenced in the HTML page has to have either absolute URLs or the <base> tag set. Both methods has advantages and drawbacks but the bottomline is that you might have to fix your templates/coding various places to be compatible.

Features

- Supports various schemes for coding the page path, including userdefined schemes
 - Pagetitles can contain spaces and characters like /.,&@ etc, the URL will still be nice.
 - URLs are generated as nice-looking lowercase paths
 - If a page is renamed, the old URL can still be used (see below in the Users Manual), so if the page was indexed by e.g. Google, it can still be found.
- Offers advanced translation of almost any set of GET parameters to/from virtual URL
 - Translation between a GET query string ("...&tx_myext[blabla]=123&type=2...") and a virtual URL (".../123/2/") is transparent to TYPO3 and all extensions; The only requirement is that the internal TYPO3 link generation functions are used ("t3lib_cObj::typolink", "t3lib_tstemplate::linkData")
- URLs are cached, so translating between URLs and IDs is very fast.
- It can handle different frames, or other pagetypes
- URLs are multilingual: if you're browsing in Dutch, you'll see Dutch URLs
- Once configured the systems works fully automatic, creating new and updating existing URLs
- You can easily see where shortcuts are pointing to, as the 'target' URL is generated, instead of the URL to the shortcut itself.
- It automatically handles installations of TYPO3 in directories other than the root of the website too
- Rawurlencoding of remaining parameter GEt vars names
- It is incompatible with "simulateStaticDocuments" .

Configuration

Installation

To install this extension, four steps must be taken:

1. Install it in the Extension Manager
2. Configure Apache / .htaccess
3. Modify your TypoScript template records with configuration for RealURL
4. Configure the extension in typo3conf/localconf.php

Install the extension

This is documented very well in the usual TYPO3 docs: just click the little gray sphere with the plus-sign and when it asks for any changes to commit, let it make them. It's not doing anything yet though.

Configure Apache

RealURLs work by providing 'virtual paths' to 'virtual files'. These don't actually exist on the file-system, so you must tell Apache to let a PHP-script handle the request if it can't find the file. This way, all URLs to pages (like www.server.com/products/product1/left.html) will be 'redirected' to /index.php, which will handle the translation of the URL into GET parameters. Real files (like images, the TYPO3 backend, static html-files, etc.) will still be handled by Apache itself though.

You should put the supplied sample .htaccess file (called `_htaccess`) in the root of your TYPO3-installation.

Alternatively, you could include the following lines in your `httpd.conf`, probably in the **VirtualHost**-section. Here is an example:

```
<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>
```

If you put it into a **.htaccess** file it has to look slightly different, basically stripping the leading slashes (“/”):

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

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

This will tell Apache that it should rewrite every URL that's not a filename, directory or symlink. It leaves everything starting with /typo3/ alone too.

Notice: For this work you need the Apache module “mod_rewrite”!

Also refer to the Appendix for extended information on mod_rewrite issues.

TypoScript configuration

Like with “simulateStaticDocuments” you need to activate the generation of the virtual file/path names in the TypoScript record – otherwise your website will not utilize the new URL encoding method.

However that is trivial; just place these four lines in the main TypoScript template record of your website:

```
0: config.simulateStaticDocuments = 0
1: config.baseURL = http://mydomain.com/
2: config.tx_realurl_enable = 1
```

Line 0 simply disables “simulateStaticDocuments” - “RealURL” is incompatible with simulateStaticDocuments and will simply not work if it has been enabled. This line should remind you of this fact.

Line 1 makes the frontend output a “<base>” tag in the header of the pages. This is required because relative references to images, stylesheets etc. will break when the virtual paths are used unless this has been set. Please see below for a detail discussion of why this is needed. Do not forget to write real name of your domain! And note slash in the end – it is required!

Line 2 enables the encoding of URLs as the virtual paths, the “Speaking URLs”.

If you use config.typoLinkEnableLinksAcrossDomains, make sure you check “Is site root?” on all pages that are site roots.

Configure the extension

Finally, you probably want to configure the way URLs are encoded. For simple needs this is quite easy and the more advanced URLs you want to encode the more configuration you need – simple, isn't it.

Configuration is done in “localconf.php” with the variable \$TYPO3_CONF_VARS[‘EXTCONF’][‘realurl’]

Please see the section later dealing with configuration options. It also offers a lot of examples.

URL en/decoding background

This section provides a bit of background information about how URLs are encoded and decoded in the system.

The general principle is that the encoding and decoding should be totally transparent to the system. This means that any extension will work with RealURL as long as they use the general link generation functions inside TYPO3 as they should do already. You can also use “simulateStaticDocuments” as a test - if it worked with that, it will (most likely) work with the RealURL extension as well.

The implementation of this transparency is done by encoding the virtual URL strictly on the basis of the GET parameters given to the encoder method. And when a HTTP request is made to a virtual URL it is decoded into a set of GET parameters which is written back to the global variables HTTP_GET_VARS / _GET - and thus any application in the system will see the parameters as if they were passed as true GET parameters.

Encoding:

URL with GET parameters -> Speaking URL -> HTML page

The encoding of the URLs happens by using a hook in the method t3lib_tstemplate::linkData(). This is configured in “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’;
```

Decoding:

HTTP request with Speaking URL -> URL is decoded, overriding values in HTTP_GET_VARS -> page rendered as always

The decoding of the URLs happens by using a hook in tslib_fe::checkAlternativeIdMethods(). This is configured like this:

```
$TYPO3_CONF_VARS[‘SC_OPTIONS’][‘tslib/class.tslib_fe.php’][‘checkAlternativeIdMethods-PostProc’][‘] =
```

```
'EXT:realurl/class.tx_realurl.php:&tx_realurl->decodeSpURL';
```

The syntax of a “Speaking URL”

Before we go on to the configuration section it's important to understand how the virtual path (Speaking URL) is decoded by the system. Lets settle an example and break it up into pieces:

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

This URL requests page id 123 with language “1” (danish) and type “1” (probably a content frame in a frameset) and on that page the display of a mininews item with id “456” is requested while the “mode” of the mininews menu is “1” (list). This parameter based URL could be translated into this Speaking URL:

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

The configuration of RealURL needed to do this magic is as follows:

```
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: );
```

In order to understand how this configuration can translate a speaking URL back to GET parameters we have to first look at how the Speaking URLs are divided into sections. This is the general syntax:

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

Each of these sections (except [fileName]) can consist of one or more *segments* divided by “/”. Thus “news/list/456/” is a *sequence* of three *segments*, namely “news”, “list” and “456”

Taking the speaking URL from above (<http://www.my-domain.dk/frontend/dk/123/news/list/456/page.html>) as an example we can now break it up into these sections:

Section	Part from Example URL	General description	Comment according to config
[TYPO3_SITE_URL]	http://www.my-domain.dk/frontend/	This part of the URL - the base URL of the site and basically where the “index.php” script of the frontend is located - is stripped of and is of no interest to the resolve of the parameters.	

Section	Part from Example URL	General description	Comment according to config
[preVars]	dk/	This section can contain from zero to any number of segments divided by "/". Each segment is bound to a GET-var by configuration in the key "preVars" (see example above) The number of segments in the pre-vars section is determined exactly by the arrays in "preVars" configuration. Typical usage would be to bind a pre-var to the "L" GET parameter so the language of the site is represented by the first segment of the virtual path.	In the configuration above there is only one pre-var configured and that is bound to the GET var "L". Further a mapping table tells that the value "dk" should be translated to a "1" for the GET var value when decoded. Also, if the segment does not match "dk" it is just ignored. meaning that the default language version of the danish page ".../dk/123/" would be just ".../123/"
[pagePath]	123/	The page path determining the page ID of the page. The default method is to just show the page ID, but through configuration you can translate say "contact/company_info/" into a page ID. The number of segments of the path used to resolve the page ID depends on the method used.	In this example the default method is used (not configured at all) and that means the raw page id (or alias if there were any) is displayed; 123. This is <i>not</i> "Speaking URL" behaviour, sorry for this weak example...
[fixedPostVars]		Fixed post vars is a sequence of fixed bindings between GET-vars and path segments, just as the pre-vars are. This is normally not useful to configure for a whole site since general parameters to pass around should probably be set as pre-vars, but you can configure fixed post vars for a single page where some application runs and in that case may come in handy. Typical usage is to apply this for a single page ID running a specific application in the system.	Not used in this example.
[postVarSets]	news/list/456/	postVarSets are sequences of GET-var bindings (in pre-var style) initiated by the first segment of the path being an identification keyword for the sequence. Decoding of postVarSets will continue until all remaining segments of the virtual path has been translated. This method can be used to effectively encode groups of GET vars (sets), typically for various plugins used on the website. Typical usage is to configure postVarSets for each plugin on the website. If postVars produce only empty segments, they will be removed completely from URLs.	In this example there is a single post var set ("news/list/456/") where the keyword "news/" (first segment) identifies the following sequence ("list/456/") to be mapped to the GET vars tx_mininews[mode] and tx_mininews[showUId] respectively.
[fileName]	page.html	The filename is always identified as the segment of the virtual path after the last slash ("/"). In the "fileName" configuration a filename can be mapped to a number of GET vars that will be set if the filename matches the index key in the array. Typical usage is to use the filename to encode the "type" or "print" GET vars of a site.	In this example the "type" value "1" is mapped to the virtual filename "page.html". In any other case the type value will be set to blank.

Configuration directives

Configuration of RealURL is done in the array \$TYPO3_CONF_VARS['EXTCONF']['realurl'] which again contains arrays. The configuration directives are broken down into these tables describing options as they are grouped together in arrays within the configuration array.

To support your understanding of the options please read the background information presented previously in this document and look at the examples available.

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

Key	Type	Description
[host-name]	->siteCfg or pointer to other key with ->siteCfg in same array	Configuration of Speaking URL coding based on current host-name for the website. Offers you the possibility of individual configuration for multiple domains in the same database. If the value of an entry is a string, then the system will expect it to point to another key in on the same level and look for ->siteCfg there. Hostname is found by t3lib_div::getIndpEnv("TYPO3_HOST_ONLY") and always in lowercase.
_DEFAULT	->siteCfg or pointer to other key with ->siteCfg in same array	Configuration of default Speaking URL coding if no matches was found for the specific HOST name.
_DOMAINS	->domainCfg	More sophisticated rewriting, bases on the value of GET variable. Typically used to convert from http://domain.com/index.php?id=12345&L=3 to http://domain.de/page-path/

Example

```

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: );

```

In this example the keys “_DEFAULT” and “www.typo3.org” is assumed to contain proper configuration of RealURL. If the hostname turns out to be “www.typo3.com” or “typo3.com” the configuration of “www.typo3.org” is used. If the hostname is “192.168.1.123” or “localhost” then the “_DEFAULT” configuration is used (which is redundant since it would be defaulted to anyways!)

->siteCfg

Key	Type	Description
init	->init	General configuration of the extension
redirects [path]	Redirect URL	Here you can specify virtual paths that should not be processed into GET vars but rather trigger a HTTP redirect header directly to the URL entered as value. If such a match happens the script issues a location-header and exits.
redirects_regex[regex]	Redirect URL	<p>Extended version of the [redirects] configuration above where you can match the path with a regex. You can also use back-references in the redirect URL.</p> <p>Example:</p> <pre>'redirects_regex' => array('^downloads/(.*)' => 'ftp://dl.domain.tld/public/dl/\1',),</pre> <p>With this configuration any URL starting with the path “downloads/” will be redirected to the ftp-server. For instance, “http://www.domain.tld/downloads/software/game.exe” will be redirected to “ftp://dl.domain.tld/public/dl/software/game.exe”</p> <pre>'^francais/(.*)' => 'fr/\1',</pre> <p>This example will redirect any URL starting with “francais/” to “fr/” which is supposedly the right key of the language on the site</p>
preVars [0..x]	->partDef	Configuration of pre-variables; a fixed set of variables bound to the initial segments of the virtual path. See description in previous section of this document.
pagePath	->pagePath	Configuration of the id-to-path transformation method. See description in previous section of this document.
fixedPostVars [pageIndex] [0..x]	->partDef	Configuration of a fixed post-variable set which does not need a keyword to trigger its interpretation. Basically like pre-vars, just set after the pagePath. See description in previous section of this document. Notice: “pageIndex” allows you to specify this for individual pages or all using “_DEFAULT” keyword. See notice below this table.
postVarSets [pageIndex] [keyword]	->postVarSet	Configuration of sets of post-variables; sets of post-variables are triggered by a keyword in the virtual path. See description in previous section of this document. Notice: “pageIndex” allows you to specify this for individual pages or all using “_DEFAULT” keyword. See notice below this table. Important: The order in which the postVarSets occur is of great importance since the first keyword definition that contains a definition for a single available GET-var will be chosen. You should arrange the postVarSets strategically.
fileName	->fileName	Configuration of filename significance; filenames can be bound to specific values of GET parameters. See description in previous section of this document.

Notice: In the table above there is defined an array key, “pageIndex”, for “fixedPostVars and postVarSets. This works mostly in the same way as the host-name pointer did for the outer level. The key can be

1. either a page id, eg. “123”

2. or the keyword “_DEFAULT”.

The value of a key should be an array (according to the definition above) but if it is a string it is interpreted as a pointer to another key on the same level.

A pointer cannot be set for “_DEFAULT”. Further, only page ids can be used (internally page aliases given as parameters will be resolved first!).

Example structure

```
1: array(  
2:   'init' => array(  
3:     ...  
4:   ),  
5:   'redirects' => array(  
6:     '' => 'cms/', // If default URL, redirect to subdir "cms/"  
7:     'test/' => 'http://www.test.test/', // If subdir is "test/" then redirect to 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: );
```

This skeleton will help you to understand the structure defined in the table above for the “->siteCfg” level in the configuration. Notice the examples for redirects.

->domainCfg

This configuration allows to append URL to the generated URL based on GET variables. It also allows to set variables back using domain names. See the example for details.

This feature requires TYPO3 4.2 or newer!

Key	Type	Description
encode	->domainEncodeCfg	Domain encoding configuration
decode	->domainDecodeCfg	Domain decoding configuration

Example

```
$TYPO3_CONF_VARS]['EXTCONF']['realurl'] = array(  
  '_DEFAULT' => array(  
    // regular default configuration  
  ),  
  'www.mywebsite.es' => array(  
    // ...  
  )  
);
```

```

        // regular configuration for a domain
    ),
    '_DOMAINS' => array(
        'encode' => array(
            array(
                'GETvar' => 'L',
                'value' => '1',
                'ifDifferentToCurrent' => true,
                'useConfiguration' => '_DEFAULT',
                'urlPrepend' => 'http://www.mywebsite.com',
            ),
            array(
                'GETvar' => 'L',
                'value' => '2',
                'ifDifferentToCurrent' => true,
                'useConfiguration' => 'www.mywebsite.es',
                'urlPrepend' => 'http://www.mywebsite.es',
            ),
        ),
        'decode' => array(
            'mywebsite.com' => array(
                'GETvars' => array(
                    'L' => '1',
                ),
                'useConfiguration' => '_DEFAULT',
            ),
            '/\..es$/' => array(
                'GETvars' => array(
                    'L' => '2',
                ),
                'useConfiguration' => 'www.mywebsite.es',
            ),
        ),
    ),
);

```

->domainEncodeCfg

Encoding configuration is an array of the following structures (arrays):

Key	Type	Description
GETvar	string	URI variable to check for encoding. Notice that this GETvar will be removed from the encoding process once this domain configuration is applied. It will be unset from the URL completely.
value	string	GETvar value for this configuration
ifDifferentToCurrent	boolean	true or false
useConfiguration	string	The name of site configuration (_DEFAULT or www.example.com)
urlPrepend	string	URL to prepend, must not end with slash

->domainDecodeCfg

Decoding configuration is an array of the following structures (arrays):

Key	Type	Description
GETvar	string	An array, where keys are URI variable names and values are values to set in URI
useConfiguration	string	The name of site configuration (_DEFAULT or www.example.com)

Each domainDecodeCfg key must be a domain name or a valid regular expression surrounded by slashes.

Do not forget www/non-www domains!

->init

General configuration of the RealURL extension

Key	Type	Description
doNotRawUrLEncodeParameterNames	boolean	<p>Disable rawurlencoding of non-translated GET parameter names during encoding.</p> <p>Background: During the encoding of Speaking URLs from GET parameters any GET parameters that cannot be translated into a Speaking URL will be set as GET parameters again. During this process the parameter name will be rawurlencoded as it actually should according to the RFCs covering this topic. This means that a parameter like "tx_myext[hello]=world" will become "tx_myext%5Bhello%5D=world" instead - which is not as nice visually but more correct technically.</p>

Key	Type	Description
enableCHashCache	boolean	If set, "cHash" GET parameters are stored in a cache table if they are the only parameters left as GET vars. This allows you to get rid of those remaining parameters that some plugins might use to enable caching of their parameter based content.
respectSimulateStaticURLs	boolean	If set, all requests where the Speaking URL path is only a single document with no path prefix (eg. "123.1.html") are ignored as Speaking URLs. This flag can enable backwards compatibility with old URLs using simulateStaticDocuments on the site. This flag is ignored if the following conditions are all met: <ul style="list-style-type: none"> ● URL does contains intermediate path segments before file name (i.e. looks like simulateStatic) ● Page ID is not found by TYPO3 core prior to realurl ● fileName.defaultToHTMLsuffixOnPrev is set to true
appendMissingSlash	boolean / string	If set, the a trailing slash will be added internally to the path if it was not set by the user. For instance someone writes "http://the.site.url/contact" with no slash in the end. "contact" will be interpreted as the filename by realurl - and the user wanted it to be the directory. So this option fixes that problem. Keyword: "ifNotFile" You can specify the option as the keyword "ifNotFile". If you use that string as value the slash gets prepended only if the last part of the path doesn't look like a filename (based on the existence of a dot "." character). Keyword: "redirect" This keyword will force RealURL to redirect to the location with appended slash. This will ensure that pages do not appear doubled in the Google index (and therefore page rank does not suffer). Notice that it is not the same as redirect action described in the ->actionConfig later in this manual. The keyword can be optionally followed by the HTTP status code to use for redirect in the square brackets (for example, "redirect[301]"). The following codes are allowed: 301, 302, 303, 307. Redirect will be processed after "ifNotFile". Keywords are separated by comma. There must be no spaces between keywords. The order of keywords in this directive is not significant.
adminJumpToBackend	boolean	If set, then the "admin" mode will not show edit icons in the frontend but rather direct the user to the backend, going directly to the page module for editing of the current page id.
postVarSet_failureMode	string (keyword)	Keyword: "redirect_goodUpperDir" . Will compose a URL from the parts successfully mapped and redirect to that. Keyword: "ignore" : A silent accept of the remaining parts. Default (blank value) is a 404 page not found from TYPO3s frontend API.
disableErrorLog	boolean	If true, 404 errors are not written to the log table.
enableUrLDecodeCache	boolean	If true, caching of URL decoding is enabled. The cache table is flushed when "all cache" is flushed in TYPO3. Entries for decode cache is valid for 24 hours by default.
enableUrLEncodeCache	boolean	If true, caching of URL encoding is enabled. The cache table is flushed when "all cache" is flushed in TYPO3.
emptyUrLReturnValue	string	If the URL is empty it usually is meant to be a link to the frontpage. If you set this value to a string, that will be the URL returned if the URL is otherwise empty. If you set this value true (PHP boolean, "TRUE"), then it will return the baseURL set in TSFE. Setting it to "." should work as a reference to the root as well. But it is not so beautiful.
reapplyAbsRefPrefix	boolean	If config.absRefPrefix is set in TypoScript setup, RealURL will strip this prefix before making the final URL. Use this option to reapply the prefix. By default it is not set, so prefix will be lost by default. If you use linking across domains, do not put full domain name in config.absRefPrefix. Use only "/" as prefix. Reapplying prefix is made after entries are written to cache.

->partDef

Definition of mapping between a segment of the virtual path and a GET variable or otherwise.

Key	Type	Description
type	string keyword	By default the array defines a GETvar mapping but there are alternatives which are configured by setting the type key: "action" : If set, the segment can define various actions, like setting frontend editing or redirecting to a certain URL, setting some parameters.
type = "action"		
index[segment] index["_DEFAULT"]	->actionConfig (or blank value which will just accept the segment but take no action on it)	The index array defines various actions and the first segment of the path is used as key to look up which action in the array to take. See ->actionConfig for more details and examples.
Default type		
GETvar		The GET var name for which this processing is done. Required The value of the GETvar will pass through a transformation defined by the other configuration options here. Basically this is the flow: <ul style="list-style-type: none"> ● First, check if "cond[prevValueInList]" allows processing and if not, return accordingly. ● The value is translated through the "valueMap" if entries matches ● If no entries in "valueMap" matched, "noMatch" is consulted for an action on this situation. ● If noMatch did not trigger, we look for a lookup table and if defined we make a look up translation ("lookUpTable"). ● If no lookup table was defined to translate the value, we look for the "valueDefault" and if set, apply that value. ● If none of these actions captured the value we just pass it through in its raw form. If "noMatch" is set to "bypass" and "valueMap" is empty, it means that variable will be erased from the generated URL on encoding and ignored on decoding.
cond[prevValueInList]	list of values	If this key is set the segment will be processed only if the previous value is found in the comma list of this value! Otherwise it will be bypassed.
valueMap	array of "segment" => "Get var value" (translation table)	The valueMap is a static translation table where each key represents a segment from the speaking URL and the value for the key is the true value of that parameter. When URLs are encoded this table is reversed and if there are duplicate values the last entry is used as speaking URL value.
noMatch	string keyword	Keyword that defines the action if the value did not match any entry in the valueMap array. "bypass" : means that if the path segment did NOT match an entry in "valueMap" the segment will be re-applied to the stack and we return/break (and thus preserves the parameter for the next segment) "null" : means that if the value is NOT found in the valuemap the getParameter is not set at all.
lookUpTable	->lookUpTable	Configuration of a database table by which to perform translation from id to alias string.

Key	Type	Description
userFunc	userFunc	<p>User function to do id<=>alias mapping. Only used if "lookUpTable" is not.</p> <p>Example configuration: <pre>'userFunc' => 'EXT:realurl/class.tx_realurl_userfunctest.php:&tx_realurl_userfunctest->main'</pre> </p> <p>Example class in RealURL (class.tx_realurl_userfunctest.php)</p> <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> <p>This class will change an id "6" to the alias "--6--" when encoded. During decode the alias "--6--" is changed back to "6". Of course this doesn't make much sense in terms of speaking URLs but with such a user function you can encode and decode ids/aliases as you like!</p>
valueDefault	string	<p>Default value to set if the path segment did not match any entries in "valueMap" or was otherwise captured for translation.</p> <p>Notice: Default values are applied AFTER any "noMatch" settings are processed (and others, see flow description for key "GETvar")</p>

Example:

```

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:    ),

```

The above example shows a configuration that allows two prevars in a path BUT they are both optional (due to the "noMatch" => "bypass" setting).

Normally a URL in the default language would look like this:

123/page.html

Then, if the L=1 GETvar is set, the URL will be like this:

danish/123/print.html

Finally, if the first segment matches "no_cache" the "no_cache=1" GET var is set and the interpretation of the language GETvar is moved to segment 2:

no_cache/danish/123/print.html

The concept of bypassing non-matched values opens the possibility of error if two values from neighbouring configurations matches. For instance errors would result from having a language labeled “no_cache” since that is a keyword in the configuration of the first segment!

Removing the “noMatch” setting will yield these URLs instead:

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

A better solution would be to set a default value for the language:

```
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:  ),
```

This would yield this result:

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

It still maintains the bypass-setting for the “no_cache” parameter but that might just fit in nicely.

Example: “fixedPostVars”

```
1:   'fixedPostVars' => array(
2:     'testPlaceholder' => array (
3:       array(
4:         'GETvar' => 'tx_extrepmgm_pi1[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_pi1[display_cat]',
17:        'valueMap' => array (
18:          'docs' => 10,
19:        ),
20:      ),
21:      array(
22:        'GETvar' => 'tx_extrepmgm_pi1[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_pi1[cmd]',
32:      )
33:    ),
34:    '1383' => 'testPlaceholder',
35:  ),
```

This configuration shows how “fixedPostVars” can be used like “preVars” but after the page path. Typically it would be used on a single page where a known plugin runs. In the above example this is the case; page id “1383” is pointed to the configuration “alias” named “testPlaceholder”. The example is designed for the typo3.org Extension Repository.

The configuration sets up a sequence of 3-4 segments in the virtual path. The first is the main menu where integer values defining the mode is mapped to nice alias strings. The second segment is the category id to display but notice how the “condPrevValue” is set to “2” - this means that *only if the previous variable was “2”* then will this segment be interpreted, otherwise bypassed! Finally there is the extension uid, here configured for translation to/from the extension keys. That is a safe process since the extension keys are unique. Finally the “command” which defines the menu level when displaying single extensions.

This configuration allows for a URL like this (4 segments in “fixedPostVars” sequence):

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

or (only 3 segments in “fixedPostVars” sequence, since first segment was not “categories” / 2)

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

->lookUpTable

Defines a table to use for look up in translation of id to alias strings for GETvars.

Key	Type	Description
table	string	Table name
id_field	string	Fieldname of the field holding the id, typically an integer, eg. “uid”
alias_field	string	Fieldname of the field holding the alias string matched to the id. Should be unique.
addWhereClause	string	Additional where clause to add to the look up query. Has to be set automatically, even for “deleted” fields since the lookup might take place before any table configuration is accessible! Example value is: “AND NOT deleted”
maxLength	integer	Defines the maximum accepted length of the alias value. If the alias value is longer than this integer the original value will be returned instead. Default value is “100”
useUniqueCache	boolean	If set, the translation from id to alias is automatically stored in a look-up table where the uniqueness of the alias is ensured; When storing it is simply checked if the alias is already associated with another ID (in the same table/fields combination) and if so a unique alias is created, typically the requested alias with numbers appended.
languageGetVar	string	Set to the GET variable name that is used to identify language uid (typically “L”). If this is set, a localized title will be looked up in case the table supports localization with “languageField” and “transOrigPointerField” set. For the configuration you HAVE TO duplicate the field names for “languageField” and “transOrigPointerField” from \$TCA of the table. This is because that during URL analysis the TCA array is not yet available to the system and thus we cannot look the values up there. An example is this, using the “tt_news” table in the modern version. See the four bold lines for an example of the combined configuration needed to make localized urls for news entries: <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	string	Same as [ctrl][languageField] in TCA for the look up table.
transOrigPointerField	string	Same as [ctrl][transOrigPointerField] in TCA for the look up table.
languageExceptionUids	string	List of sys_language uids to except in case “languageGetVar” is used (see below).
useUniqueCache_conf['strtolower']	boolean	If set, the aliases are converted strtolower()
useUniqueCache_conf['spaceCharacter']	string	Normally, this defaults to an underscore (_), which is used to replace spaces and such in an URL. You can set this to e.g. a hyphen (-) if you want to.
useUniqueCache_conf['encodeTitle_userProc']	userFunc	Additional processing of the alias before it is cached.
enable404forInvalidAlias	boolean	If true, a 404 will be thrown if an alias being resolved wasn't found to match and id.

Key	Type	Description
autoUpdate	boolean	If true, (and useUniqueCache is set) the aliases will automatically update themselves.
expireDays	integer	Number of days in which older aliases will expire if "autoUpdate" is used. Default is 60 days.

Example

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

->actionConfig

Key	Type	Description
type	string keyword	<p>The action is identified by one of these keywords:</p> <p>“admin” : Enables the Frontend Editing features (similar feature to ->postVarSet, type = “admin”).</p> <p>“redirect” : Redirects you to another URL with optional markers filled with the value of the path segment and the remaining path. Useful for site shortcuts</p> <p>“notfound” : Throws a 404 error</p> <p>“bypass” : Bypass AND adds the key to the stack again! (because the key belongs to next part of URL)</p> <p>“feLogin” : Adds keyword when users are logged in. Feature that allows different URL during logins, mainly useful for cache-control headers which needs a different URL for logged in sessions in order to server non-cached pages for login users. Notice: This feature will actually check for a logged in user; groups added by IP masks or otherwise will not be recognized as a login session.</p> <p>default: Passthrough(<i>without</i> adding key to stack).</p> <p>Notice about the “_DEFAULT” key: If the “_DEFAULT” type is <i>not</i> “bypass” (meaning that the _DEFAULT action is to <i>do</i> something) the URL encoding method will look for the first occurrence of an action of no type (passthrough without adding key to stack) and use that as segment value.</p>
Only type “redirect”		
url	string	<p>The URL to redirect to. There are two markers you can use in the URL:</p> <p>####INDEX### - Inserts the current segment.</p> <p>####REMAIN_PATH### - Inserts the remaining path from this point (including any filename)</p> <p>The values are rawurlencoded()</p>

Example: Redirects and required prefixes

```
1: 'redirects' => array(
2:     '' => 'cms/',
3:     'mailinglist/' => 'http://lists.netfielders.de',
4: ),
5: 'preVars' => array (
6:     array(
7:         'type' => 'action',           // "type" action
8:         'index' => array(
9:             'cms' => '',           // Just bypass
10:            'admin' => array(
11:                'type' => 'admin'   // Jump BE login OR setting frontend edit flags...
12:            ),
13:            'search' => array(
14:                'type' => 'redirect', // Redirect...
15:                'url' => 'index.php?id=1344&tx_indexedsearch[sword]=####REMAIN_PATH###',
16:            ),
17:            'ext' => array(
18:                'type' => 'redirect', // Redirect...
19:                'url' => 'cms/1383/list/####REMAIN_PATH###/index.html',
20:            ),
21:            '_DEFAULT' => array(
22:                'type' => 'notfound' // If key was not found in index, throw "404" not found.
23:            ),
24:        ),
25:    ),
26: ),
```

In this example the the first segment of the URL is configured to be an action. The segment is required since the “_DEFAULT” index is set to “notfound” meaning that if none of the other keys are matched you will see a “Page not found” error.

Anyways, the example takes offset in the typo3.org website and this is the consequences of the above configuration:

URL	What happens
http://typo3.org/	This URL would result in a 404 error if it wasn't for the redirect configuration in line 1-4 of the configuration. Here the system is ordered to redirect to “cms/” in case there is no virtual path found.
http://typo3.org/cms/1420/	This will show page ID 1420. The action key “cms” doesn't do anything - it just bypasses the processing to the next level which is the page ID resolving. By configuring such a prefix you basically define all of your site to “run” from the virtual directory “cms/”. Configuration is in line 9
http://typo3.org/admin/1420/	This will also show page ID 1420 but activate the frontend editing icons in the interface - and if the user is not currently logged in as a backend user a redirect to the backend login form will happen. Configuration of this feature is in line 10-12 When using the Frontend Editing trigger (“admin”) in RealURL the “admin/” segment of the path will be carried around in the links on the site so you stay in admin mode until you remove it again. But this is also means that caching is disabled for all pages so page links are never stored with the “admin/” action in!
http://typo3.org/search/system+requirements	Will redirect to http://typo3.org/index.php?id=1344&tx_indexedsearch[sword]=system%2Brequirement which is the page where the indexed_search plugin is running and automatically performing a search for the words after “search”. This is configured in line 13-16; notice how the search word (the remaining path) is automatically inserted in the URL by the marker string “###REMAIN_PATH###”
http://typo3.org/ext/lang	Will redirect to http://typo3.org/cms/1383/list/lang/index.html where “lang” is inserted by the marker “###REMAIN_PATH###” just like with the “search” action before. The principles are the same. Configuration in line 17-20
http://typo3.org/maillinglist/	Will redirect to http://lists.netfielders.de according to line 3 in the configuration

Example: Language prefix and “admin” action

```

1:     'preVars' => array (
2:         array(
3:             'type' => 'action',           // "type" action
4:             'index' => array(
5:                 'admin' => array(
6:                     'type' => 'admin'     // Jump BE login OR setting frontend edit flags...
7:                 ),
8:                 'search' => array(
9:                     'type' => 'redirect',   // Redirect...
10:                    'url' => 'index.php?id=1344&tx_indexedsearch[sword]=###REMAIN_PATH###',
11:                ),
12:                'ext' => array(
13:                    'type' => 'redirect',   // Redirect...
14:                    'url' => 'cms/1383/list/###REMAIN_PATH###/index.html',
15:                ),
16:                '_DEFAULT' => array(
17:                    'type' => 'bypass'     // If key was not found in index, throw "404" not found.
18:                ),
19:            ),
20:        ),
21:        array(
22:            'GETvar' => 'L',
23:            'valueMap' => array(
24:                'dk' => '1',
25:            ),
26:            'noMatch' => 'bypass',
27:        ),
28:    ),

```

In this example two preVars are configured, the first is an action containing almost the same actions as the previous example except that the “_DEFAULT” configuration is of the “bypass” type which means that if none of the key matches the first segment the interpreter will simply move on to the next preVar configuration for that segment (bypassing and adding segment value to stack again).

In addition there is a language prefix configured as well.

The result of this configuration should be clear from looking at the examples in the following table:

URL	What happens
http://typo3.org/1420/	Shows page 1420
http://typo3.org/admin/1420/	Shows page 1420 with Frontend Editing icons

URL	What happens
http://typo3.org/dk/1420/	Shows page 1420 in danish (&L=1)
http://typo3.org/admin/dk/1420/	Shows page 1420 in danish (&L=1) with Frontend Editing icons

Example: feLogin passthrough

This configuration will make “loginarea/” a prefix in all URLs used when a frontend user is logged in. In itself the prefix does nothing; it sets no parameter internally. But it can be crucial for cache-control because it allows you to send cache-headers to the client browser for all pages where no user is logged in while all login pages are associated with another URL (the main URL prefixed with this keyword) for which you can set no-cache headers.

Example configuration:

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

->pagePath

Configuration of the method that en/decodes the id to/from a “page path”

Key	Type	Description
type	string	Setting the method used for encoding/decoding of the ID The default simply is to set the page id/alias as a single entry in the virtual path. “user” : Calls external class for generation.
Only type “user”:		
userFunc	function reference	Function reference to handle the id encoding. Examples can be found in class.tx_realurl_dummy.php An full fledged implementation is found in “class.tx_realurl_advanced.php”. See more details later in documentation about this. Example value: EXT:realurl/class.tx_realurl_advanced.php:&tx_realurl_advanced->main
rootpage_id	integer	Defines the root page uid of the site for which the configuration is made. Required. See example in the tx_realurl_advanced section.
[other keys can depend on user functions.]		

Example of page id translation to path:

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

Calls a user function which will render a true Speaking URL of the page titles and not just output the page id numbers. See a thorough description of this class later in this document, chapter “class.tx_realurl_advanced.php”

By the configuration above URLs will look like this (before / after):

```
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/

Example of dummy setup:

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

Calls a dummy class which does exactly what the main class does: Outputs the page id/alias and nothing more. But if you want to implement your own schemes this class is a useful offset for you!

->postVarSet

Key	Type	Description
type	string keyword	<p>By default a postVarSet consists of</p> <ul style="list-style-type: none"> ● a keyword that identifies the following sequence in the virtual path ● a sequence of one or more segments in the path which is mapped to GETvars <p>An example (from previously) is “news/list/456/” where the keyword “news/” (first segment) identifies the following sequence (“list/456/”) to be mapped to the GET-vars tx_mininews[mode] and tx_mininews[showUid] respectively (according to configuration). The configuration of the sequence is done by a numeric array of ->partDef.</p> <p>Other modes: There are other modes than the default mode and they can be triggered by setting the “type” key to one of the following keywords. In these alternative cases there might not be any sequence of segments after the keyword, but still the keyword is triggering the alternative mode so that will always be around.</p> <p>“single” : Using this keyword you bind the keyword to represent an exact amount of GETvars with exact values. This works precisely like filenames are bound to GETvars (see ->fileName)</p> <p>“admin” : Using this keyword the backend will enable frontend editing mode for the user, showing the context sensitive edit icons in the frontend. If the user is not logged in as a backend user there will be a redirect to the backend login form where the user can login and after successful login the user will be redirected to the previous page.</p>
Only default type:		
[0..x]	->partDef	The configuration of the sequence associated with the “keyword” that defines this postVarSet.
Only “single” type:		
keyValues	array of [GETvar] => [string value]	The “keyValues” array defines one or more GET variables with <i>specific</i> values. The keyword of the postVarSet is matched if if <i>all</i> the GET-vars configured in [“keyValues”] is found in the remaining pool of GET vars that needs translations <i>and</i> if the values match exactly!

Example: Frontend edit

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

Adding lines 4-6 in the above codesnippet to the postVarSets of a configuration will enable frontend edit mode if users append “.../edit_now/” to the virtual path. Of course you can choose any “admin-directory” you like.

One warning here; If the keyword is appended to a URL where a previous postVarSet sequence is not yet finished then the keyword will of course be seen as a parameter of that postVarSet and *not* as the keyword triggering the frontend edit mode as you wanted. Therefore you might want to use the same feature but for pre-vars instead.

Example: PostVarSets

```
1:     'postVarSets' => array(
2:         '_DEFAULT' => array (
3:             'plaintext' => array(
4:                 'type' => 'single',    // Special feature of 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:     ),
```

This example shows how three sets of postVarSets has been configured of which two of them are the default type (keyword + sequence of GETvars) while the third is of the “single” type, mapping to a fixed GETvar value.

In order to understand this configuration and the effect it has you should study these commented examples based on the above configuration. Each example consists of two lines where the first is the URL with GETparameters and the second is the Speaking URL version of the first.

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

Above, the postVarSet “ext” is used to encode the GET parameters. The sequence is initiated by the keyword “ext” and the following three segments of the virtual path is mapped to the three GET-vars configured for the keyword and in the order they appear in the configuration above (line 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/
```

The above two examples shows what happens if only one or two of the parameters are used - basically the empty values are stripped off from the *end* of the path. The first example would actually render “123/ext/aaa//” and the second would be “123/ext/aaa/bbb//” but since the empty values are in the end of the path we can safely strip them off as the example shows.

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

In this example only “tx_myExt[p1]” and “tx_myExt[p3]” is used and since the sequence requires “p2” to be in between we have to accept the empty segment of the virtual path.

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

In the above example the mininews parameters are encoded, using the keyword “news”. Notice that the “tx_mininews[mode]” GETvar has a mapping table which allows automated translation between the value “1” and “list” used in the virtual URL. This feature (and other similar options) allows to create truly speaking URLs even for parameters that are ID numbers.

```
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/
```

In this example we have two postVarSets, namely “ext” and “news”. As you can see this is no problem at all as long as the sequences contains the correct amount of segments so the next keyword gets registered.

Notice that the “ext” keyword gets listed first. This is because the “ext” postVarSet is the first one found in the configuration and therefore is triggered before the “news” postVarSet.

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

In this example we have left out two parameters from the previous URL and that means there are two empty segments in the virtual path. There is no way around this since the sequence length has to be respected and in case of the “news” postVarSet the “mode” was defined before the “showUId” parameter which means it is not possible to strip of the the empty value.

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

This example is only different to the above examples by showing what happens to an unknown GET var when a URL is encoded; quite simply that GET var is appended to the final URL - what else!

->fileName

Configuration of the significance of the filename in the virtual path.

Key	Type	Description
index[filename][“keyValues”]	array of [GETvar] => [string value]	The “index” is an array of virtual filenames (eg. “page.html”) which is associated with one or more GET variables with <i>specific</i> values. A filename is chosen during encoding if <i>all</i> the GET-vars configured in index[filename][“keyValues”] is found in the remaining pool of GET vars that needs translations <i>and</i> if the values match exactly! The filename value “_DEFAULT” is used if no match was found. Important: The order in which the filenames occur is of great importance since the first filename that matches will be chosen. You should arrange the filenames strategically. See example below.
defaultToHTMLsuffixOnPrev	boolean/string	If set, then the last directory part of the virtual path being made will be turned into the filename suffixed “.html” IF the filename part is non-existing. For example, “workplace-learning-solutions/companion-solutions/” would be turned into “workplace-learning-solutions/companion-solutions.html” and the basepart of the filename (stripping the “.html” extension) will still be perceived as the last part of the virtual path. This approach is useful if you want to simulate HTML documents even if you don’t configure any fileName mappings. If set to string, that string will be used as suffix. Notice that leading dot is mandatory (i.e. valid suffix is “.html”, not just “html”)
acceptHTMLsuffix	boolean/string	If URL contains “.html” suffix and this property is enabled, then this suffix will be stripped from url. This allows you to easily migrate old site to TYPO3, keep all incoming external links working but use urls with another suffix (through defaultToHTMLsuffixOnPrevid is) or without suffix at all.

Example: Multiple filenames for a 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:     ),
24:     'acceptHTMLsuffix' => '.cfm'

```

This example could be configuration for a frames-based website. When there should be no “type” value the default key is used (which renders no filename of course) but if the &type value is 1 or 2 either “page.html” or “top.html” is used; those filenames will then represent the GET var “&type=1” and “&type=2” respectively during decoding.

Also notice how the “&print=1” parameter has been encoded into a filename! The idea is that if the filename is “print.html” then two GET vars are set; both “&type=1” and “&print=1”. *But you must be very careful* how you arrange the filenames; if “print.html” was entered below “page.html” then it would never be used since the first match wins and “page.html” would be found to match exactly with “&type=1” and thus the “&print=1” GET var would be appended the URL (like “page.html?print=1”) instead of being encoded into the filename (“print.html”).

Example: Default filename

```

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

```

This example shows a configuration that will prefix the filename “index.html” by default no matter what.

Handling relative links with Speaking URLs

By default, TYPO3 generates all links to other pages as www.server.com/index.php?id=123&type=0, so all pages seem to be in one (filesystem-) directory: the root of the website. The problem is, that many extensions (and TYPO3 core code) rely on images, javascripts, etc. to be in a directory relative to the TYPO3-root, like “typo3/ext/indexed_search/pi/res/pages.gif”. This approach doesn’t work when the path is constantly changing.

For example, a file “fileadmin/my_image.jpg” referenced from “index.php?id=123” will be found because “index.php” is in the root of the website where also the “fileadmin/” folder is. But as soon as the URL “index.php?id=123” is encoded into a Speaking URL, say “contact/company_address/”, then your browser will try to find the image in “contact/company_address/fileadmin/my_image.jpg” where is obviously isn’t located.

So to solve this problem you

1. either have to prefix all relative references with an absolute path to the site root
2. or set the <base> tag in the HTML files header to the site root.

config.absRefPrefix

There is a TypoScript-setup directive to set an absolute prefix to all links and images (config.absRefPrefix), but sadly enough that isn’t implemented in all places (the indexed-search and front-end-editing for example), so that doesn’t work too well.

Please don’t use config.absRefPrefix. It has some nasty properties that render RealURLs complete unusable sometimes. The only problem is that the 404-page of TYPO3 doesn’t have the <base>-tag, so it doesn’t show the TYPO3-logo :)

Support for this might be allowed when the bugs are fixed but generally it will require all code generating reference to use this method and that cannot be guaranteed for all extensions of course.

<base> tag

There is a very simple solution in HTML though: just supply the <base>-tag in the <head> of your pages, like:

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

To make your TypoScript templates RealURL-enabled, you should therefore include this statement in your HTML-templates, or use the following TypoScript snippet:

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

This will automatically read what the current base URL is on your website (using `t3lib_div::getIndpEnv('TYPO3_SITE_URL')`) and create a `<base>` tag in the header of the HTML output in the frontend.

The `<base>` tag method seems to work flawlessly in TYPO3 except in two cases where you have a link like `` - this will *not* work because it refers to the site root but obviously is an 'internal' reference in the current document.

However you can solve this situation in a simple way:

```
config.prefixLocalAnchors = all
```

This will set the needed prefix for all occurrences of '`<a href="#"...."....' in the page; basically anywhere a local anchor is generated. This substitution happens by a ereg_replace on the general page content after rendering. See TSref for details.`

The other situation is specific for MSIE; When you set the "document.location" via JavaScript, MSIE will understand relative URLs as relative to the current URL, not the `<base>` URL. Therefore you will have to prefix the base URL. You can find that value in `$GLOBALS['TSFE']->baseUrl` (or use `t3lib_div::getIndpEnv("TYPO3_SITE_URL")`).

Making extensions compatible with “config.baseURL”

If you set “`config.baseURL`” and subsequently “`config.prefixLocalAnchors = all`” then extensions might still produce wrong local anchors. That is if extensions are including un-cached page content by `USER_INT` or `USER_EXT` cObjects that content is *not* processed! (unless “`config.prefixLocalAnchors`” is set to “output”). For such extensions there should be inherent support for RealURL and that can be done (with full backwards compatibility) by prefixing all local anchors made by the result of this:

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

or in recent TYPO3 versions:

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

Generally

Make sure you include either configuration it in ALL page-types that are generated!

class.tx_realurl_advanced.php

Introduction

The class tx_realurl_advanced offers advanced encoding of page IDs to paths including encoding in localized titles and cache management.

Configuration

You should create Domain-records on the pages where domains start. Even if you only have one domain, it's a good idea to create a Domain-record for it. There's one thing you should note:

If you have installed TYPO3 in the document-root of a host, you should create a domain-record named like www.server.com. If, on the other hand, your TYPO3-installation is in a different directory, you should create a domain-record named something like www.server.com/the_path_to_your_typosite. Slashes at the end don't matter that much.

Configure RealURL to work with “tx_realurl_advanced” ID encoding

Simply set this configuration for the key “pagePath” in the configuration array:

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

The directives specific for “tx_realurl_advanced” are these:

Directive:	Data Type:	Description:
languageGetVar	string	Defines which GET variable in the URL that defines language id; if set the path will take this language value into account and try to generate the path in localized version.
languageExceptionUids	string	Comma list of UIDs of sys_language records which will not generate a localized URL if languageGetVar is set.
spaceCharacter		Normally, this defaults to an underscore (_), which is used to replace spaces and such in an URL. You can set this to e.g. a hyphen (-) if you want to.
segTitleFieldList	list of fieldnames	The prioritized order of field names from pages table (<i>root line !</i>) that is used when building the speaking URL. Default is “tx_realurl_pathsegment,alias,nav_title,title” (for Alternative Page Language records this is only “nav_title, title”). Notice: If you specify user defined fields which are not currently in root line you will have to add them to the root line via “\$GLOBALS['TYPO3_CONF_VARS']['FE']['addRootLineFields]”
disablePathCache	boolean	Will disable the usage of path cache. The system will rely solely on forward-lookups on the fields in “segTitleFieldList”.
autoUpdatePathCache	boolean	(Depends on “disablePathCache” being false!) If set, the URLs will automatically update themselves after changes to the page title, alias and other fields specified in “segTitleFieldList” and keep track of older values for a period of time corresponding to “expireDays” (see below). The cost of this feature is that the pathCache is not used for speedy lookups but used for tracking these URLs instead. Thus, performance wise it corresponds to setting “disablePathCache” to zero.
expireDays	integer	The time the old URL of a page whose pagetitle changed will still be remembered (in days). See “autoUpdatePathCache” above. Default is 60 days.
firstHitPathCache	boolean	If set, then the path-cache look up is accepted only if it returns a result in the first hit. In versions prior to 1.4 this will prevent “was not set as postVarSet as expected” error but at a cost of extra database lookups. Since version 1.4 it is not longer needed to set this option to prevent that error but option is kept for compatibility reasons.
rootpage_id	integer	This setting is critical for proper functioning of tx_realurl_advanced in the multi domain environment. Its description can be found in ->pagePath section. In versions prior to 1.0.1 it was specific to tx_realurl_advanced but now it is a realurl setting. See example below.
encodeTitle_userProc	user function	Additional user processing in “encodeTitle()”.
dontResolveShortcuts	boolean	If set, page shortcuts are not resolved to their destination page. NB: If you do not set this option the backends Speaking URL Management module will probably report duplicate entries in the pathCache table since both the shortcut page and the destination page will have the same URL entry. The error is in that case not a real problem of course, but annoying to the eye.
excludePageIds	string	Comma list of page ids to exclude from being RealURL rendered. The list must NOT contain any spaces between page id numbers!

Example: Configuration of RealURL for use on more than one domain in the same database

```

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: );

```

Notice how the rootpage_id field is set differently for these two cases!

Hooks

Starting from version 1.5.3 RealURL contains hooks to allow extra processing of URLs. Here is a list of hooks.

encodeSpURL_postProc

This hook is called before the RealURL returns created URL to the caller. This allows extensions to modify URL in the way they want. The hook receives an array with parameters:

Parameter:	Data Type:	Description:
pObj	tx_realurl	A reference to the calling object
params	array	Parameters passed to the encodeSpURL
URL	string	Generated URL. This is a reference, so modifying this variable will modify RealURL result

decodeSpURL_preProc

This hook is called before the RealURL starts URL decoding. This allows extensions to modify URL in the way they want. The hook receives an array with parameters:

Parameter:	Data Type:	Description:
pObj	tx_realurl	A reference to the calling object
params	array	Parameters passed to the decodeSpURL
URL	string	URL to determine id for. This is a reference, so modifying this variable will modify URL that RealURL processes

Automatic configuration

Overview

Since version 1.4 RealURL has automatic configuration option. It checks the system and attempts to write configuration file to simplify work for user. Such check is done if user did not specify configuration manually. RealURL tries to make optimal configuration and it does so for most common cases. If your host needs specific settings, you should either write configuration manually or modify generated configuration.

RealURL will create configuration for each domain and for each language defined in the system. To create language configuration it requires `static_info_tables` extension. `static_info_tables` is suggested during installation of RealURL. If you do not plan to use automatic configuration or you have only one language, you can skip `static_info_tables`.

RealURL may store generated configuration in one of two formats: PHP serialized array or source PHP code. These options are offered during extension installation. The first option is at least 10 times faster (may be less with PHP accelerator) and it should be used in production environment. However serialized arrays cannot (and must not be under any conditions!) modified manually. Serialized array may stop working after you upgrade PHP version, though this happens really rare. In any case, use the second option only if:

- you plan to modify generated configuration
- you plan to update to newer PHP version really soon

Important: configuration is generated only once. If you add domain or new language to the system, automatic configuration will not be updated. You have to delete a file in `typo3conf/` directory named `realurl_autoconf.php`. RealURL will regenerate configuration if this file does not exist. Note that `realurl_autoconf.php` is not the same as `realurl_conf.php` or any other similar named file. It is always `realurl_autoconf.php` and not something else.

If you use manual configuration, you can disable autogeneration by clearing a checkbox while installing extension. This will save you a couple of milliseconds.

Automatic configuration of extensions

Extensions can alter generated RealURL configuration to suit their needs. Notice that it is done only once when configuration is generated. If you install an extension that supports autogeneration, you need to delete a file in `typo3conf/` directory named `realurl_autoconf.php`. RealURL will regenerate configuration if this file does not exist.

To add/modify RealURL configuration, extensions must provide a hook to RealURL. This hook will be called once and receive RealURL configuration as array. This array is a *template*, not a final configuration. RealURL will use this array to generate separate set of arrays for each domain. Think about this array as about `_DEFAULT` entry in RealURL configuration.

The following code example from `album3x` extension shows how to set such hook in `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'
```

Here is an example of the hook from `album3x` extension:

```
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_pi1[page]',
                    ),
                ),
            'image3x' => array(
                array(
                    'GETvar' =>
'tx_album3x_pi1[showUid]',
                    'userFunc' =>
'EXT:album3x/class.tx_album3x_realurl.php:&tx_album3x_realurl->main',
                ),
            ),
        ),
    ));
}

```

Important note: try to invent preVar, postVar and fixedPostVar names that do not conflict with other extensions. Since variable names should look good in the URL, choice is limited and conflicts may occur. For example, mininews extension try to prevent conflict with tt_news by changing postVar name if tt_news is installed:

```

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_pi1[showUid]'
                    ),
                ),
            ),
        ));
    }
}

```

Appendix

mod_rewrite notice from Ole Tange, www.fi.dk

When using RealUrl you will need a rewrite rule in .htaccess (or apache.conf) that rewrites the path to index.php. This will do that:

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

However, Apache should not rewrite documents located in /typo3/, /uploads/, /fileadmin/ and /typo3conf/:

```

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

```

Also /typo3 (without trailing '/') should not be rewritten:

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

If you have a PDF-file that you want to give a nice URL (e.g. www.example.com/project-name/report.pdf) you would want to put it in [/project-name/](http://www.example.com/project-name/). Therefore files that do exist should not be rewritten either:

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

If you use a symlink to the file then this should not be rewritten either:

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

If www.example.com/project-name is not a file then it should be considered as a directory so that a relative link to 'background' will be a link to www.example.com/project-name/background and not to www.example.com/background

To force this you simply append a '/' to the url if it is not a file:

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

This works by redirecting the browser from www.example.com/project-name to www.example.com/project-name/. Note that it will break `rss.xml` and similar mappings in the `fileName` section of the configuration.

If in the directory `project-name` there is an `index.html` we should use that instead of `Typo3`:

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

The same goes for `index.htm` and `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]
```

The resulting `.htaccess` looks like this:

```
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
```

Please note that the order is important. Changing the order will most likely give undesired results.

Using `httpd.conf` for setting it up, a tip from Stefan Bühler (copy-paste from bugtracker)

I now have a solution for httpd.conf:

```
-----  
RewriteRule ^/(typo3|typo3temp|typo3conf|t3lib|t3lib|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]  
-----
```

This sets the environment variable "ORIG_SCRIPT_NAME" to "/index.php" which Typo3 uses (if present) instead of "SCRIPT_NAME".
"SCRIPT_NAME" is only changed to this if you use .htaccess with RewriteBase, and it is not possible (as far as i know and tried) to change this from httpd.conf.
If your Typo3 is reachable e.g. at http://host/path-to-typo3/, [^] you would change the RewriteRule to:
RewriteRule ^/(.*)\$ /index.php [L,E=ORIG_SCRIPT_NAME:/path-to-typo3/index.php]

ToDo list

- For technical todo list see "doc/TODO.txt" in the extension

- Some clever 404 page can be created, using the builtin indexed-search for example. The requested language (/nl/...) could help to provide the page in the requested language. Nothing done with this yet though.

Changelog

- 0.0.1: First upload
- 0.0.4: First working version
- 0.0.5: An icon was added, thanks to Netcreators for creating it!
- 0.0.6: Documentation in StarOffice format added
- 0.0.7: Almost complete rewrite / revision of the code, implemented 'path_to_typo'-feature, implemented support for multiple domains, changed the code so that most of the configuration is now automatic, updated documentation
- 0.1.0: First publicly available version
- 0.1.1: pathPrefix didn't work correctly, so a hack was added to allow it too work now
- 0.1.2: URLs in pages weren't rendered correctly on single-language-sites
- 0.1.3: Added possibility to choose another character to replace a space (instead of an underscore), fixed another stupid bug regarding the rendering of some URLs
- TOTAL re-organization by Kasper Skårhøj march 2004
- See Changelog file inside extension (standard CVS changelog)