LifeTips
Move Up In Life.

White Paper

# Maintaining URLs in Search Engine Indexes

Aaron Robinson | Director of Technology | LifeTips.com

LifeTips.com
One 1st Avenue
Building 34
Suite 200
Charlestown, MA 02129
617.886.9001

## General URL Maintenance

A well planned URL maintenance strategy is a vital part of any website content migration.  If any URL from your site will no longer be available, a suitable replacement should be found.  The continued rendering of your pages is something that both the search engines and your daily visitors expect.  Also, each unique piece of content should only have one location on your website.

There are two steps to rewriting URLs:

- Exposing the new page URL (if you are changing the location of the page) throughout your website

- Handling requests for the old hyperlink by issuing a 301 redirect to the client along with the new location of the page.

By removing references to the old page, you're helping to stop any future users from requesting a defunct URL.  Also, as search engines walk the web, they add links they come across into an index of pages that they will visit in the future to parse for content.

By returning a 301 redirect for the old page, you keep the user from getting a 404 error, as well as help search engines to keep their indexes clean of content they've discovered at different URLs that they may see as duplicate content.  Another reason to implement 301 redirects for your old URLs is that many other sites may have linked to your site using them, and these inbound links are helping your position in the search engines.  If all of these inbound links begin serving 404 errors, then the value provided by those links is thrown out.

## Implementing Friendly URLs

Friendly URLs, where English language words are embedded into the page address, can help your website in several ways.  First and foremost is that it allows you to expose more metadata to search engines that spider your site.  Another important aspect is that the user that finds your page in those search results sees those same values in their search results, and this may help them to choose your page over other results.

A friendly URL scheme can be implemented easily on a static site by naming the files appropriately and having one file per page.  For larger sites that may be database-driven, this ceases to be an option.  Maintenance of 1000's of .html files would be infeasible.

Instead, a rewriting layer should be provided above dynamic pages to translate a seemingly static URL into something the web server can have processed by a server-side script.  The search engines and users see an .html file, but the server ends up processing something in .asp, .php, or any other script host.

In most cases, implementation of a rewriting layer is done well after an initial site has gone live.  These sites have been indexed with their query string parameter based pages.  These pages will need to be removed from any search engine indexes and replaced with the new friendly URL.

The twist to this implementation is that these aren't new pages.  They are simply new addresses for your old pages.  A request for the original URL can still be received by the web server and processed normally.

To properly implement friendly URLs on an existing site, there are three steps:

- Exposing the new page URL throughout your website

- Handling requests for the old hyperlink by issuing a 301 redirect to the client along with the new public location of the page.

- Rewrite requests for your new URL scheme, turning it back into what the web server software can process in its old manner.

Once again, by removing references on your site to the old page addresses, you're helping the search engine spiders to keep their indexes clean.  You're also helping end users to browse your site faster, as well as reducing load on your server, as you will forever serve 301 redirects for the old URL if you don't remove references of it from your website.

By serving a 301 redirect for your old URLs, you keep the search engine index clean of duplicate content and get your new page addresses listed in the search engines instead.  Eventually, the redirects you provide should only be for other sites that link to you, where you do not control their content.  Traffic for the old URL should eventually approach nothing.

Finally, you should implement a mapping layer to translate friendly URLs into unfriendly URLs.  This is what allows you to make very little changes on your website while still providing strong metadata in your URL structure.  When an incoming request to the web server is made, the web server has an opportunity to change the request before passing it off to the executable that processes the request.  It can take this opportunity to parse and create a new URL for the server, possibly executing an entirely different page from what the client requested, without a round trip to the client, or the knowledge of the client.  A request is made, a response is provided.

## Exposing a new URL

To expose your new URL, you need to simply replace all references of hyperlinks in your old codebase with the new address.  In large sites data-driven sites, helper functions to create these URLs can reduce this migration work.  The use of server-side includes or templates in your codebase can help to reduce the amount of time it takes to change the static portions of your website.

After your site goes live with the new URL structure, you can monitor the log files your server creates from inbound client requests, taking note of any requests for pages on your website that show a referring page that is also on your website.  This will allow you to track down any errant links on your site that need to be translated.

## Redirection

*There are many ways to redirect users of your website to a new location:*

- *Refresh META tag, sent in the head section of an HTML document, with additional content possibly telling the user where the new location is and providing a link if their browser doesn't perform the redirect automatically.*

- *302 Redirect, indicating a temporary redirect from an old URL to a placeholder URL*
- *301 Redirect, indicating a permanent redirect from an old URL to a new URL*
- *Client-side script, likely JavaScript, that tell the browser to change its location*

Of these, the 301 redirect is the best bet when dealing with users and search engine spiders.  It tells the client that the new location is a permanent location and that it should update any local references or indexes for that address to this new location.  The SEO impact of this is that the search engine can transfer indexed value of the old page to the new one.

The Refresh META tag can also be used where a 301 redirect cannot.  Its implementation by search engines is not as well defined, with different search engines taking different stances on how its "time to refresh" value translates to the semantics of a 301 or 302 redirect.  If this method is used, you should certainly provide information in the body section of the HTML document, as many browsers allow you to turn these refresh directives off.

Client-side script should not be used for redirects.  Many users turn off script in their browsers for security reasons, and search engines do not parse and execute client-side script, so using this in a URL rewriting strategy will not work.

## Rewriting URLs

Depending on the web server platform you are running, there are different options for rewriting requests, as well as several different patterns that can be used when implementing this technology. When running under IIS on Windows, there is a free utility named ISAPIRewrite that can be installed that will allow you to configure, using regular expressions, mapping of URLs from old to new. A similar layer exists on Apache in Mod_Rewrite, that allows for regular expression matching of an old URL and rewriting this into a new URL. Both of these technologies allow the pass-through of a new URL to the web server for transparent processing, as well as the ability to immediately redirect to a new URL, sending a 301 or 302 redirect back to the client without the need to load the page processing code.

In database-driven sites, you will want to include an identifier that allows you to map to an appropriate page, passing all the information that is needed to lookup the record in the database and server the response. Any extra friendly values in the URL, such as product name, are extra values that are unnecessary for the proper processing of the page. Part of what you will want to do when loading the target page is to check that the incoming request was made with the current canonical value for this extra bit of information. While it isn't used by the page to serve the request, you want to be consistent in what the public URL for any page is. If you update the product name, URLs that are circulating with the old product name can be 301 redirected by the server code to the new URL. The client will receive this redirection notice and request the page with the latest proper URL.

## Implementation Examples

When friendly URLs were first implemented at LifeTips, we referred to our category-level pages across all of our Tip Sites using:

```
/Cat.asp?id=12345
```

There were many problems with this URL structure. The first, and most serious, is that it uses an 'id' parameter in the URL, which Google clearly states they do not index, making the assumption that it is a session identifier. The other large problem with the URL is that it does not provide any Meta data as to what you might find on that page. Other issues are that we're telling the world what scripting platform we're using to process our pages and that we're using query string parameters.

Our new URL structure is what you can see currently on the LifeTips site:

```
/cat/12345/global-warming/index.html
```

When ISAPIRewrite intercepts the inbound request to the web server, we can pattern match that string into various pieces.

When ISAPIRewrite intercepts the inbound request to the web server, we can pattern match that string into various pieces.

The first piece in the URL pattern is the first pseudo subdirectory, cat, which helps us to direct our rewriting pattern to the one we have in place to point to the Cat.asp script on our web server.

The second piece is the original database identifier for that category in our database. These are the two required pieces of any data-driven friendly URL implementation.

The third piece of the URL is a normalized version of the category name.  We've lowercased it, replaced any special characters with dashes, and replaced any whitespace with dashes as well.  Dashes are used because search engines see a dash as a word barrier, while an underscore (or no delimiter at all, running the words together) doesn't break the words into distinct pieces.

The 'index.html' was then tacked onto the end, again not providing much value to the solution, but as something we liked to have.  It could certainly have been left off and the URL rewriting effort would have worked out just as well.

Anytime we write out a page that references this category page, we call into a helper method that takes the pattern type, the database identifier, and the extra string.  The resulting string is written as the URL rather than the Cat.asp link we used to provide.

We then provided a redirect from the old page to the new page.  To do this, we created a common page that would perform this work of looking up a database entry, creating the canonical URL, and sending a 301 redirect.  We directed inbound requests for Cat.asp using the ISAPIRewrite layer, creating query string parameters for the common page that tell it what type of request we needed a redirection for, as well as the ID to use in its lookup process:

```
RewriteRule /Cat.asp\?id\= (\d*) /301Redirect.asp\?page=cat\
&id=$1 [I,O]
```

This provides our 301Redirect.asp script with everything it needs to send a 301 redirect to the client with the new URL.

The final step to migrating to friendly URLs is to map the friendly URL back to the original unfriendly URL so that the server script does not need to be modified.  The regular expression pattern we used in this case was:

```
RewriteRule /cat/(\d*)/([^/]*)/index.html /Cat.asp\?id=$1\
&Category=$2 [I,O]
```

This pattern looks for 'cat', followed by a forward slash, any number of digits representing the database identifier, followed by a forward slash and any string, representing the category name, followed by a forward slash and the literal 'index.html'. The server will continue processing of the request by executing the Cat.asp page, unbeknownst to the client.

The parentheses in the expression allow for references to be made in the new URL. As you can see, we pass the first match using $1 into the id query string parameter, and the second match using $2 for the Category parameter. The reason we pass in the Category here, where it did not exist in our original URL structure, is so that the page can compare this value to the current database value and redirect the user if they have requested a version of the page for this database identifier with a different name value. This is the magic that lets you update a product name and still have a user or search engine end up at the page with the latest correct version of the URL. In our example, if we update the category name from 'Global Warming' to 'Effects of Global Warming', the Cat.asp script would receive 'global-warming' as the query string parameter, compare this to the current expected value of 'effects-of-global-warming', and send a 301 redirect response to the client telling them of the new page location.

## Conclusion

By making small changes to a sites HTML, combined with a dose of redirection and rewriting, you can keep users and search engines from requesting pages on your site that no longer exist, instead sending them to where you think they should be on your site.