Google code

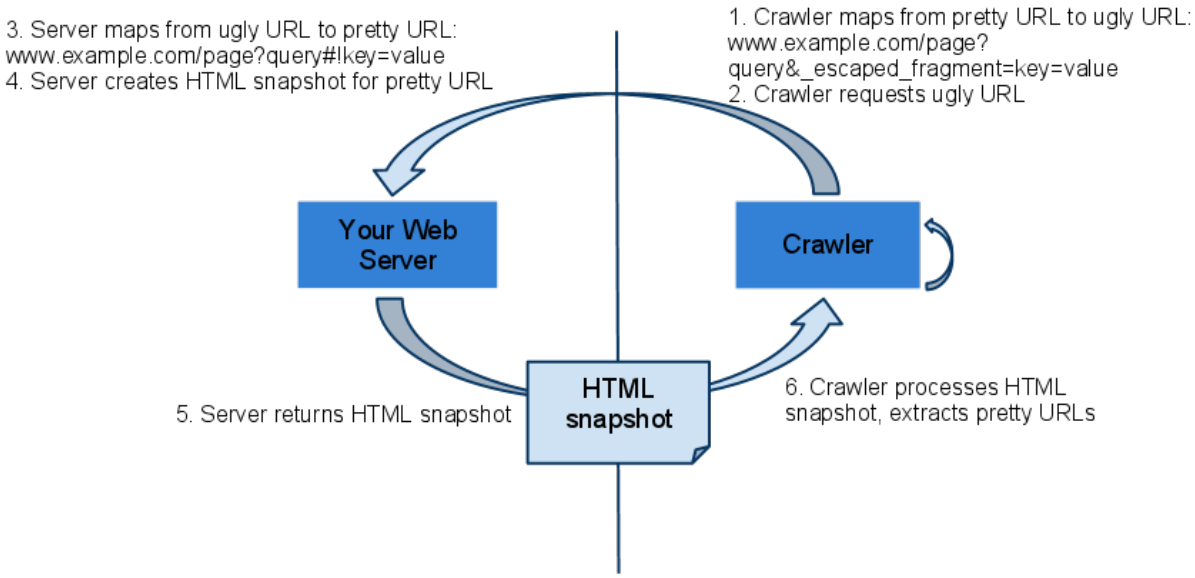
Making AJAX Applications Crawlable

Getting Started

This document outlines the steps that are necessary in order to make your AJAX application crawlable. Once you have fully understood each of these steps, it should not take you very long to actually make your application crawlable! However, you do need to understand each of the steps involved, so we recommend reading this guide in its entirety.

Overview of Solution

Briefly, the solution works as follows: the crawler finds a pretty AJAX URL (that is, a URL containing a `#!` hash fragment). It then requests the content for this URL from your server in a slightly modified form. Your web server returns the content in the form of an HTML snapshot, which is then processed by the crawler. The search results will show the original URL.



Step-by-step guide

1. Indicate to the crawler that your site supports the AJAX crawling scheme

The first step to getting your AJAX site indexed is to indicate to the crawler that your site supports the AJAX crawling scheme. The way to do this is to use a special token in your hash fragments (that is, everything after the # sign in a URL): hash fragments have to begin with an exclamation mark. For example, if your AJAX app contains a URL like this:

```
www.example.com/ajax.html#key=value
```

it should now become this:

```
www.example.com/ajax.html#!key=value
```

When your site adopts the scheme, it will be considered "AJAX crawlable." This means that the crawler will see the content of your app if your site supplies HTML snapshots.

2. Set up your server to handle requests for URLs that contain `_escaped_fragment_`

Suppose you would like to get `www.example.com/index.html#!key=value` indexed. Your part of the agreement is to provide the crawler with an HTML snapshot of this URL, so that the crawler sees the content. How will your server know when to return an HTML snapshot instead of a regular page? The answer is the URL that is requested by the crawler: the crawler will modify each AJAX URL such as

```
www.example.com/ajax.html#!key=value
```

to temporarily become

```
www.example.com/ajax.html?_escaped_fragment_=key=value
```

You may wonder why this is necessary. There are two very important reasons:

- Hash fragments are never (by specification) sent to the server as part of an HTTP request. In other words, the crawler needs some way to let your server know that it wants the content for the URL `www.example.com/ajax.html#!key=value` (as opposed to simply `www.example.com/ajax.html`).
- Your server, on the other hand, needs to know that it has to return an HTML snapshot, rather than the normal page sent to the browser. Remember: an HTML snapshot is all the content that appears on the page after the JavaScript has been executed. Your server's end of the agreement is to return the HTML snapshot for `www.example.com/index.html#!key=value` (that is, the original URL!) to the crawler.

Note: The crawler escapes certain characters in the fragment during the transformation. To retrieve the original fragment, make sure to unescape all %XX characters in the fragment. More specifically, %26 should become &, %20 should become a space, %23 should become #, and %25 should become %, and so on.

Now that you have your original URL back and you know what content the crawler is requesting, you need to produce an HTML snapshot. How do you do that? There are various ways; here are some of them:

- If a lot of your content is produced with JavaScript, you may want to use a headless browser such as HtmlUnit to obtain the HTML snapshot. Alternatively, you can use a different tool such as crawljax or watij.com.
- If much of your content is produced with a server-side technology such as PHP or ASP.NET, you can use your existing code and only replace the JavaScript portions of your web page with static or server-side created HTML.
- You can create a static version of your pages offline, as is the current practice. For example, many applications draw content from a database that is then rendered by the browser. Instead, you may create a separate HTML page for each AJAX URL.

It's highly recommended that you try out your HTML snapshot mechanism. It's important to make sure that the headless browser indeed renders the content of your application's state correctly. Surely you'll want to know what the crawler will see, right? To do this, you can write a small test application and see the output, or you can use a tool such as Fetch as Googlebot.

To summarize, make sure the following happens on your server:

- A request URL of the form `www.example.com/ajax.html?_escaped_fragment_=key=value` is mapped back to its original form: `www.example.com/ajax.html#!key=value`.
- The token is URL unescaped. The easiest way to do this is to use standard URL decoding. For example, in Java you would do this:

```
mydecodedfragment = URLDecoder.decode(myencodedfragment, "UTF-8");
```

- An HTML snapshot is returned, ideally along with a prominent link at the top of the page, letting end users know that they have reached the `_escaped_fragment_` URL in error. (Remember that `_escaped_fragment_` URLs are meant to be used only by crawlers.) For all requests that do not have an `_escaped_fragment_`, the server will return content as before.

3. Handle pages without hash fragments

Some of your pages may not have hash fragments. For example, you might want your home page to be `www.example.com`, rather than `www.example.com#!home`. For this reason, we have a special provision for pages without hash fragments.

Note: Make sure you use this option only for pages that contain dynamic, Ajax-created content. For pages that have only static content, it would not give extra information to the crawler, but it would put extra load on your and Google's servers.

In order to make pages without hash fragments crawlable, you include a special meta tag in the head of the HTML of your page. The meta tag takes the following form:

```
<meta name="fragment" content="!">
```

This indicates to the crawler that it should crawl the ugly version of this URL. As per the above agreement, the crawler will temporarily map the pretty URL to the corresponding ugly URL. In other words, if you place `<meta name="fragment" content="!">` into the page `www.example.com`, the crawler will temporarily map this URL to `www.example.com?_escaped_fragment_=` and will request this from your server. Your server should then return the HTML snapshot corresponding to `www.example.com`. Please note that one important restriction applies to this meta tag: the only valid content is "!". In other words, the meta tag will always take the exact form: `<meta name="fragment" content="!">`, which indicates an empty hash fragment, but a page with AJAX content.

4. Consider updating your Sitemap to list the new AJAX URLs

Crawlers use Sitemaps to complement their discovery crawl. Your Sitemap should include the version of your URLs that you'd prefer to have displayed in search results, so in most cases it would be `http://example.com/ajax.html#!key=value`. **Do not include links such as `http://example.com/ajax.html?_escaped_fragment_=key=value` in the Sitemap. Googlebot does not follow links that contain `_escaped_fragment_`!** If you have an entry page to your site, such as your homepage, that you would like displayed in search results without the `#!`, then add this URL to the Sitemap as is. For instance, if you want this version displayed in search results:

```
http://example.com/
```

then include

```
http://example.com/
```

in your Sitemap and make sure that `<meta name="fragment" content="!">` is included in the head of the HTML document. For more information, check out our additional articles on Sitemaps.

5. Optionally, but importantly, test the crawlability of your app: see what the crawler sees with "Fetch as Googlebot".

Google provides a tool that will allow you to get an idea of what the crawler sees, Fetch as Googlebot. You should use this tool to see whether your implementation is correct and whether the bot can now see all the content you want a user to see. It is also important to use this tool to ensure that your site is not cloaking.