Content Negotiation: why it is useful, and how to make it work

21 February 2006 by Olivier Thereaux | Posted in: HTTP, Technology 101

We recently received a puzzled message from a visitor of the W3C Web site, asking how we were serving images without file suffix in their URI. Looking around, our visitor found that `http://www.w3.org/StyleSheets/TR/logo-REC` was not one file, but two: `logo-REC.gif` and `logo-REC.png`. How do we do that?

The short answer is: *Content Negotiation*.

In this article, we will discuss content negotiation in depth and examine practical solutions. However, to begin, we need to first understand what a URI is, and what it is not.

A URI is a reference

The first thing we need to understand is that **a URI is not a file name**.

It is convenient to see a URI as the location for a file, and in most cases, the analogy works. However, as we will see soon, this analogy is too poor to describe everything a URI actually is. Let's just remember that there is a good reasons why web "pedants" insist on calling it a URI, a *Universal Resource Identifier*, not a URL: it is not a file name or location, it is an *identifier (or a reference) to a resource*. By using a proper protocol, it is possible to retrieve the actual resource, that's called *dereferencing a URI*.

But why all this abstraction, since in most cases the resource will happen to be stored in a file anyway, and the URI will be mapped directly to the file name?

Let us consider two things very similar to a URI: a bar code for a product, and an ISBN for a book. The former is a reference to a product, and the latter, a reference to a publication. In the case of the bar code and associated product, it is important to note that the product is not a specific box of cookies on a shelf, the referred product is actually the type of cookies of a certain brand, and all share the same bar code. Similarly, ISBNs do not refer to a flesh and bone (or, rather, paper-and-spine) book, but to the text it contains. In fact, it is not rare that several editions of a book share the same ISBN number: in the context of the ISBN, they are similar.

The same idea can be applied to URIs. A URI refers to a resource, but the resource is not one file on one web server. Take for example the resource "the weather in Oaxaca". A resource is just that: a piece of information on the Web. An HTML document with a text describing the weather in Oaxaca, or an image representing a map with indicators about the weather, all these files can be appropriate representations for this resource.

In fact, the maintainer of the Web resource could very well decide that a number of representations of this piece of information are equivalent, and think "what if I let the visitors of my Web site decide which representation they prefer?" On the Web, these equivalent representations of a resource are called *variants*, and the mechanism used to determine which of the existing representations is most appropriate for a given request is called *Content Negotiation*.

Content Negotiation: figuring out the best deal for everyone

Content Negotiation is a complex-sounding term for what is a rather simple mechanism.

Imagine yourself discussing on the phone, suggesting a date. You ask: We should meet soon! How about Wednesday, or Friday?. your friend answers Excellent! I have free time on Friday!. Sounds simple? Now replace the *date* with a *resource*, think of yourself as the *client requesting a resource* and your friend as the *Server, accepting the request based on the preferences of the client and on its own availabilities*: this is Content-Negotiation as it is implemented on the Web.

In summary, the basic idea of Content Negotiation is to serve the best variant for a resource, and to serve it based on:

- What variants are available, and what variants the server may prefer to serve
- What the client can accept, and with which preferences: in HTTP, this is done by the client which may send, in its request, *Accept* headers (Accept, Accept-Language and Accept-Encoding), to communicate its capabilities and preferences in Format, Language and Encoding, respectively.

<!– this part on server-driven vs client driven was not really useful, taking it oot

By default, the HTTP protocol implements what is called a *server-driven* content negotiation mechanism, meaning that the Web server, upon receiving information about the Client's supported variants and preferences, as well as knowing the available variants for a resource, will be the actor of the negotiation responsible for making the final decision on which variant suits everyone best. The opposite is *client-driven* negotiation, where the server lists all variants, and asks the client: "pick one". The latter is sometimes used as a fallback mechanism when the server-driven mechanism fails.

–>

Language Negotiation: why every multilingual site owner should know about it

The mechanism that allows us to serve an image in two different file formats, which our visitor was puzzled about, is in fact one type of Content-Negotiation, called *Format Negotiation*. One other important and interesting usage of Content Negotiation is its application to representations of a resource in several languages, and how to serve them to the reader based on their preferences: *Language Negotiation*.

With Language Negotiation, there is no need to give a link to oaxaca.html.en for readers of English and oaxaca.html.de for readers of German, just link to oaxaca, set up your server properly (e.g apache) and the negotiation happening between the server and the client's preference will make each reader receive the resource in the proper language.

Why is Language Negotiation seldom used?

How come then that language negotiation is not being widely used at all if it can be so useful in dispatching, automatically, the proper language variant of a document to its audience? Partly perhaps because it is not well known, and people building multilingual web sites think of their site as a multiplication of language-specific mini-sites, instead of thinking of it as one site, with one set of URIs, only with different versions and languages available.

It is not, however, the sole reason for the lack of usage of language negotiation. One other reason is that for a long time, with the most popular negotiation-enabled Web server (the ubiquitous apache), failed negotiation (for instance, a reader of french being proposed only english and german variants of a document), resulted in a nasty “406 not acceptable” HTTP error, which, while technically conforming to HTTP, failed to follow the recommendation that a server should try to serve some resource rather than an error message, whenever possible. Fortunately, more recent versions of the server now allow the setting of a fallback, or default, variant in case the negotiation fails.

Another serious issue: giving the *users*, not the *browser*, what they want

There is another issue with language negotiation as it is implemented in HTTP: it implies that the client is properly configured, that is, it implies that the client (the Web browser) will send `Accept-Language` information that actually reflects the languages its user can read, and what languages are preferred among these. Unfortunately, it is often not true: although many modern browsers do allow their users to set preferred languages, not all of them do, and even when they do, there are cases when the user does not know how the set up is made (here is how). In some cases, for instance on shared computers or “internet kiosks”, the user is not even allowed to change the settings of the Web browser.

In this context, a zealous usage of language negotiation can even have effects against usability of a site. Imagine a bilingual site (in our example, English and Japanese) where negotiation between the server and browser results in the choice of the English variant. The reader actually prefers Japanese, and finds a link to the Japanese version, easily visible at the top of the English variant, and follows it. However, as the user keeps browsing… the negotiation between the browser and the server keeps returning the English version. Quite probably, the user will just get irritated, browse away, never to return: language negotiation, albeit there to help the user, can prove to be a usability liability.

Toward a *better* language negotiation

How can we work around this?

One possibility is to choose to provide “generic”, language negotiated access to resources only at known important entry points to the site, and from there on, use only language specific links . That solution does prevent the running away of users irritated by the limitations of language negotiation, but if Bob wants to send a link to a specific resource on the site to his friend Norio in Japan, wouldn’t it be nice to be able to just send the URI of the page he is browsing (in English) and have his Japanese friend automatically get the Japanese version? Wouldn’t it be nice to be able to use the power of Language negotiation on the whole site, without any usability issue?

After all, the concept of negotiation is to try and automatically provide the best possible variant based on the ones available in the server and the preferences of the user – whether to use the preferences of the browser and the `Accept` HTTP headers it sends is only a convenient implementation in HTTP, not the *only* way to implement a negotiation system.

What if?… Negotiation could take into account the settings in the user’s browsers, *and* records of past interactions with the site. And although HTTP is stateless, there is an easy way to do this: cookies. A negotiation algorithm trusting a cookie showing that the user has *chosen* a language different than the one negotiated based on its Accept-Language: header information, and defaulting to Accept-based negotiation in the absence of such

a cookie, may be the best of both worlds: negotiated resources, and the guarantee of a consistent user experience regardless of potentially misconfigured browsers.

A PHP implementation of the “better language negotiation”

Below is a sample implementation of the idea described above, using the php language.

How this php-based language-negotiation works

1. `page` (the URI naming is just an example) is the “generic” resource. It checks for the existence of a language choice cookie first, and in the absence of it, calls `choose_lang.php`. When the negotiation algorithm ends, a variable called `$chosenlang` is set, and based on the value of this variable, either `page.en` or `page.ja` is called with an `include` mechanism

   You can download the php source for `page`. Its code is actually very simple, as shown below:

   ```
   <?php
   include('/path/to/choose_lang.php');
   if ($chosenlang == "en") {
           include 'page.en';
   }
   else {
           include 'page.ja';
   }
   ?>
   ```

2. `choose_lang.php` implements a very basic HTTP language negotiation based on Accept-Language: headers. It does not take into account “quality factors”, which could be used to weigh in several possible choices. As you can see in the commented source, its main task is to find a value for the variable `$chosenlang`, first by checking the presence of a cookie (denoting a language choice in previous interaction with the site), then by trying a content-negotiation algorithm similar to that of HTTP, and finally, if necessary, falling back to a default language choice.

3. Finally, the language specific files for our page, `page.en` and `page.ja` should have some code at the top, executed only if the variable `$chosenlang` is not set. As we saw above, this would mean that the resource was *not* called through the generic resource, but rather requested directly, so it’s a fair assumption that the user followed a language-specific link to switch the language of display: therefore, we want to store a cookie recording the new choice of language.

   Here is how the code at the top of `page.en` should look:

   ```
   <?php if(! isset($chosenlang)) {setcookie("lang", "en", time()+60*60*24*30, "/"); $chosenlang="en";} ?>
   <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   …
   ```

Links galore

- HTTP1.1: Content Negotiation. *The* reference. There is also an RFC on Content Negotiation in HTTP, but it's a little outdated, and refers to an now superseded draft of HTTP1.1
- The short and sweet definition of Content Negotiation from the Web architecture document.
- the CHIPs W3C Note has a number of guidelines on how Web servers and server-side technologies can implement negotiation.
- The Internationalization Activity of W3C has two good FAQs on the topic of language negotiation: When to use it and How to set it up with Apache.
- The always thorough Jukka Korpela has a extensive section of his site dedicated to multilingual Web sites, discussing technical as well as usability aspects.
- The bilingual Tokyo Art Beat Web site uses the language negotiation techniques explained in this article.

Post Scriptum

Many thanks to Karl Dubost, Felix Sasaki and Steph Troeth for some excellent input, suggestions and corrections to this article.

Please use the comments form below if you wish to provide feedback, or suggest other implementations to make Content Negotiation on the Web more useful and more widely used. Thank you.

Post navigation

← Buy standards compliant Web sites | Blog home | Minutes of QA IG F2F at the W3C Tech Plenary – February 2006 →

17 thoughts on "Content Negotiation: why it is useful, and how to make it work"

Malte Steckmeister says:
February 21, 2006 at 12:30 am

When dealing with content-negotiation, I always stumble upon the following question:
Doesn't a user expext a certain language when browsing to a resource on a server with country-domain regardless of client-language-accept-settings?
If a user goes to http://www.domain.de, I think he/she would expect de (german) content and would be irritated when the server gives back en content.
Especially because many many clients out there are misconfigured and do not reflect the actual language preferences/capabilities of the user.

Motaz says:
February 21, 2006 at 5:46 am

I think using cookie only to save the language is a bad idea !

It will be much better to use cookies + session and $_GET query string to let search engens read all of your pages in all of your lagnuages

Regards

Frankie Roberto says:
March 2, 2006 at 5:30 am

A nice article, and I agree that using content-negotiation plus cookies is a good way to go at key entry points to your site. However I think of cases where sending a URL to your friend and having them see the page in a different language might be completely confusing. There may also be cases where you want to temporarily view a page in a different language from the one you usually prefer.

One neat solution might be to have a 'generic' URI for each page, which uses content negotiation, plus a language-specific one which 'forces' a resource to be displayed a given language. I'm not sure whether this would be best set as a sub-domain (http://en.site/page), a file exension (http://site/page.en) or a query string parameter (http://site/page?lang=en) though.

You could then set up your site so that search engines index the language-specific versions.

olivier says:
March 28, 2006 at 8:42 am

Malte: that's an excellent point, although I would tend to disagree. When I go to e.g http://www.verybigcompany.co.jp, I expect it to be the site of the verybigcompany in Japan, but if they serve me the content in a language I specifically prefer to Japanese, I'm a happy customer.

Motaz: could you give details on why you disagree with the idea of using cookies to store language preferences? I would also be interested in hearing argument defending session-related query strings in URIs (preferred to cookies), which I would tend to think as ugly, unnecessary, not to mention, often a security hazard.

Frankie: I agree completely. I suspect the article failed to state clearly that it's a good usability practice to generally link to the language-negotiated resource, but that the resource itself should have links to its alternatives in other languages. I will try to add that somehow.

Thanks for your comments and suggestions.

Kanashii says:
April 1, 2006 at 5:49 pm

I argee with Frankie's suggestion of using sub-domains to serve alternative languages.

All links to the site will work without having to modify them for various languages (e.g. /link/), the default language can be served without the sub-domain giving a clean and simple URI, and the copying of links will represent the language originally viewed in.

Though it wouldn't be practical for everyone, it seems to be the best of all worlds without having to rely on cookies or query strings.

Fabio Pinna says:
June 27, 2006 at 1:11 pm

There's a easy solution to that.

Step 0: implement any Language Negotiation method (the PHP one described in this page is nice, albeit it needs some modification imho)

Step 1: implement a $_GET method for specifying a language preference.
http://www.mysite.com/index.php?lang=xx is good.

Step 2: mod_rewrite. And set it up to translate http://www.mysite.com/xx/index.php to http://www.mysite.com/index.php?lang=xx

This method is

1. scalable (you can add as many languages as you want via a map for mod_rewrite and some PHP coding)
2. efficient (mod_rewrite settings can be built directly into apache configuration files)
3. generates nice URIs
4. d) completely replaceable with some (maybe "a lot of") PHP/ASP coding for those servers that do not support mod_rewrite or something similar.

If an user calls a normal URI, then we do Language Negotiation to provide him the appropriate content. If he chooses a specific language, we provide him the link to the appropriate URI, that will in turn provide the appropriate content.

I can see that this method defeats the purpose of having a single set of URIs for each page: we have 1 + x URIs, where x is the number of languages we support. It, however, also defeats the need for cookies (which I think are a bad idea in most cases), and enables the user to bookmark his language preference, thus making bookmark exchanging possible.

One final note: if an user calls for a language we don't support, we issue a 301 redirect (Permanently Moved) to the "generic" page, that then does Language Negotiation as usual. This also accounts for

search engines, and allows them to index our site in every language (because we put links on the language-specific pages).

Just my two cents.

Fabio Pinna says:
June 28, 2006 at 10:22 am

Oh, I almost forgot: of course, we can also mod_rewrite a subdomain: http://xx.mysite.com to http://www.mysite.com/index.php?lang=xx is an example.

Nikolay Bagrov says:
September 7, 2006 at 12:17 pm

Not all sites can use mod*rewrite for subdomains. In this case domain owner must set nonstandard configuration for DNS. If owner have an access to virtual or real server it work. But if he buy hosting with control panel without shell access often he can't modify DNS records for use subdomains without previous declare.*
*In this case best way is use SESSION ans COOKIES (may be with mod*rewrite) but not subdomains.
Almost this way easy if you port your system to another OS or change hoster. In last case you must declare all of domains new.
Language selection is a simple. More hard be create a control pannel with many users. You can use subdomains for each. But most simple way – use subdirs with mod_rewrite and SESSIONs. In this case you can scale your system to some servers easy, for example.

Just my five cents. :)

Irfan says:
January 5, 2011 at 5:21 am

Hi,
I have a question. Why does the website have an extension of en or ja? I know that corresponds to name but is it also the entension as well? Because I do not believe .en os a valid file extension.

Or is the complete name of the page in English page.en.php and for Japanese page.ja.php?

Thanks in Advance,
Irfan.

Coralie Mercier says:
January 5, 2011 at 9:18 am

> Irfan, no, it isn't the extension. It is what you call your resource, be that a php page of an html page, and how you can refer to it with content negotiation.

Irfan mir says:
January 5, 2011 at 11:43 pm

> Well, hmm I'm a little confused still. In the code below if the language is English, it includes page.en what is page.en? Is it a php page whose content is in English?
>
> And if if not English is page.ja a php page in Japanese?
>
> ```
> <?php
> include('/path/to/choose_lang.php');
> if ($chosenlang == "en") {
> include 'page.en';
> }
> else {
> include 'page.ja';
> }
> ?>
> ```
>
> well what I am confused on is that it is including a page with content in a specific language that it is in the same directory as the php code above is in?
>
> Is there anything incorrect in that scentence?
> I apologise if I am not clear, but I was just wondering what file/document do I include?
>
> Thank you for all your help and regards,
> Irfan mir

Coralie Mercier says:
January 6, 2011 at 4:00 pm

> Irfan, I don't think I can explain differently the whole content of the section [[A PHP implementation of the "better language negotiation"]] in the article above.
>
> The point is to include as many documents as you have languages, in this case two, English and Japanese.

Irfan says:
January 7, 2011 at 4:36 am

Language negotiation

I apologise, yes it would be absurd for you to personally explain the section. However, could you check if I wrote this code properly and that it is valid?

In page.php, it would have:

```
<?php
include('php/choose_lang.php');
if ($chosenlang == "en") {
    include 'languages/page/english.php';
}
else {
    include 'languages/page/spanish.php';
}
?>
```

then this would be in english.php:

```
<?php if(! isset($chosenlang)) {setcookie("lang", "en", time()+60*60*24*30, "/"); $chosenlang="en";} ?>
<!DOCTYPE HTML>
<HTML lang="en">
  <head>
    <meta charset="utf-8″>
     <title> Page in English </title>
  </head>
  <body>
      <p> I am written in English </p>
  </body>
</HTML>
```

And this in spanish.php

```
<?php if(! isset($chosenlang)) {setcookie("lang", "es", time()+60*60*24*30, "/"); $chosenlang="es";} ?>
<!DOCTYPE HTML>
<HTML lang="es">
  <head>
    <meta charset="utf-8″>
     <title> Page in Spanish </title>
  </head>
  <body>
      <p> se escritó en español </p>
  </body>
</HTML>
```

Does this code look good /proper ?

Thank you for all your help & Best Regards :-),
Irfan

Coralie Mercier says:
January 7, 2011 at 8:38 am

Irfan, I have no idea but hopefully you'll find an answer in relevant fora.
Best regards,
Coralie

Luis Oscar says:
May 11, 2011 at 5:46 am

I have the same unanswered question that were asked to you. How are you doing image content negotiation? (The PHP code) I am interested for a mobile site in which I preffer the png over gif; but png is not as widely used by mobile browsers. I also want to pass W3C MobileOK Checker while serving png images.

Dominique Hazael-Massieux says:
May 11, 2011 at 7:44 am

For content negotiation using PHP, you need to analyze the "Accept" header and determine which MIME type is preferred by the browser.

There are various methods to do that, with varying level of robustness. I have personally successfully make use of the Content Negotation class at http://www.w3.org/2005/04/conneg.phi.

It works for instance with the following code:

```
$conneg = new contentNegotiation();
if ($conneg->compareQ("image/png,image/gif")=="image/png") {
// Serve PNG image
} else {
// Serve GIF image
}
```

Note that you can learn more about content negotiation (in particular for mobile devices) in the on-line course W3C is running: http://www.w3.org/Mobile/training/

Hope this helps,
Dom

Mable Kyung says:
November 16, 2011 at 6:07 am

I'm trying to use the latest beta you linked to with PHP 5.3.1, but no luck. The extension is not loaded. Did you put any APC related settings to your php.ini, because my php.ini has no settings for it?

Comments are closed.