Apache httpd Tutorial: Introduction to Server Side Includes

Server-side includes provide a means to add dynamic content to existing HTML documents.

- Introduction
- What are SSI?
- Configuring your server to permit SSI
- Basic SSI directives
- Additional examples
- What else can I config?
- Executing commands
- Advanced SSI techniques
- Conclusion

See also
- Comments

Introduction

Related Modules	Related Directives
mod_include	Options
mod_cgi	XBitHack
mod_expires	AddType
	SetOutputFilter
	BrowserMatchNoCase

This article deals with Server Side Includes, usually called simply SSI. In this article, I'll talk about configuring your server to permit SSI, and introduce some basic SSI techniques for adding dynamic content to your existing HTML pages.

In the latter part of the article, we'll talk about some of the somewhat more advanced things that can be done with SSI, such as conditional statements in your SSI directives.

What are SSI?

SSI (Server Side Includes) are directives that are placed in HTML pages, and evaluated on the server while the pages are being served. They let you add dynamically generated content to an existing HTML page, without having to serve the entire page via a CGI program, or other dynamic technology.

For example, you might place a directive into an existing HTML page, such as:

```
<!--#echo var="DATE_LOCAL" -->
```

And, when the page is served, this fragment will be evaluated and replaced with its value:

```
Tuesday, 15-Jan-2013 19:28:54 EST
```

The decision of when to use SSI, and when to have your page entirely generated by some program, is usually a matter of how much of the page is static, and how much needs to be recalculated every time the page is served. SSI is a great way to add small pieces of information, such as the current time - shown above. But if a majority of your page is being generated at the time that it is served, you need to look for some other solution.

Configuring your server to permit SSI

To permit SSI on your server, you must have the following directive either in your `httpd.conf` file, or in a `.htaccess` file:

```
Options +Includes
```

This tells Apache that you want to permit files to be parsed for SSI directives. Note that most configurations contain multiple `Options` directives that can override each other. You will probably need to apply the `Options` to the specific directory where you want SSI enabled in order to assure that it gets evaluated last.

Not just any file is parsed for SSI directives. You have to tell Apache which files should be parsed. There are two ways to do this. You can tell Apache to parse any file with a particular file extension, such as `.shtml`, with the following directives:

```
AddType text/html .shtml
AddOutputFilter INCLUDES .shtml
```

One disadvantage to this approach is that if you wanted to add SSI directives to an existing page, you would have to change the name of that page, and all links to that page, in order to give it a `.shtml` extension, so that those directives would be executed.

The other method is to use the `XBitHack` directive:

```
XBitHack on
```

`XBitHack` tells Apache to parse files for SSI directives if they have the execute bit set. So, to add SSI directives to an existing page, rather than having to change the file name, you would just need to make the file executable using `chmod`.

```
chmod +x pagename.html
```

A brief comment about what not to do. You'll occasionally see people recommending that you just tell Apache to parse all `.html` files for SSI, so that you don't have to mess with `.shtml` file names. These folks have perhaps not heard about `XBitHack`. The thing to keep in mind is that, by doing this, you're requiring that Apache read through every single file that it sends out to clients, even if they don't contain any SSI directives. This can slow things down quite a bit, and is not a good idea.

Of course, on Windows, there is no such thing as an execute bit to set, so that limits your options a little.

In its default configuration, Apache does not send the last modified date or content length HTTP headers on SSI pages, because these values are difficult to calculate for dynamic content. This can prevent your document from being cached, and result in slower perceived client performance. There are two ways to solve this:

1. Use the `XBitHack Full` configuration. This tells Apache to determine the last modified date by looking only at the date of the originally requested file, ignoring the modification date of any included files.
2. Use the directives provided by `mod_expires` to set an explicit expiration time on your files, thereby letting browsers and proxies know that it is acceptable to cache them.

Basic SSI directives

SSI directives have the following syntax:

```
<!--#function attribute=value attribute=value ... -->
```

It is formatted like an HTML comment, so if you don't have SSI correctly enabled, the browser will ignore it, but it will still be visible in the HTML source. If you have SSI correctly configured, the directive will be replaced with its results.

The function can be one of a number of things, and we'll talk some more about most of these in the next installment of this series. For now, here are some examples of what you can do with SSI

Today's date

```
<!--#echo var="DATE_LOCAL" -->
```

The `echo` function just spits out the value of a variable. There are a number of standard variables, which include the whole set of environment variables that are available to CGI programs. Also, you can define your own variables with the `set` function.

If you don't like the format in which the date gets printed, you can use the `config` function, with a `timefmt` attribute, to modify that formatting.

```
<!--#config timefmt="%A %B %d, %Y" -->
Today is <!--#echo var="DATE_LOCAL" -->
```

Modification date of the file

```
This document last modified <!--#flastmod
file="index.html" -->
```

This function is also subject to `timefmt` format configurations.

Including the results of a CGI program

This is one of the more common uses of SSI - to output the results of a CGI program, such as everybody's favorite, a ``hit counter."

```
<!--#include virtual="/cgi-bin/counter.pl" -->
```

Additional examples

Following are some specific examples of things you can do in your HTML documents with SSI.

When was this document modified?

Earlier, we mentioned that you could use SSI to inform the user when the document was most recently modified. However, the actual method for doing that was left somewhat in question. The following code, placed in your HTML document, will put such a time stamp on your page. Of course, you will have to have SSI correctly enabled, as discussed above.

```
<!--#config timefmt="%A %B %d, %Y" -->
This file last modified <!--#flastmod file="ssi.shtml" --
>
```

Of course, you will need to replace the `ssi.shtml` with the actual name of the file that you're referring to. This can be inconvenient if you're just looking for a generic piece of code that you can paste into any file, so you probably want to use the `LAST_MODIFIED` variable instead:

```
<!--#config timefmt="%D" -->
This file last modified <!--#echo var="LAST_MODIFIED" -->
```

For more details on the `timefmt` format, go to your favorite search site and look for `strftime`. The syntax is the same.

Including a standard footer

If you are managing any site that is more than a few pages, you may find that making changes to all those pages can be a real pain, particularly if you are trying to maintain some kind of standard look across all those pages.

Using an include file for a header and/or a footer can reduce the burden of these updates. You just have to make one footer file, and then include it into each page with the `include` SSI command. The `include` function can determine what file to include with either the `file` attribute, or the `virtual` attribute. The `file` attribute is a file path, *relative to the current directory*. That means that it cannot be an absolute file path (starting with /), nor can it contain ../ as part of that path. The `virtual` attribute is probably more useful, and should specify a URL relative to the document being served. It can start with a /, but must be on the same server as the file being served.

```
<!--#include virtual="/footer.html" -->
```

I'll frequently combine the last two things, putting a `LAST_MODIFIED` directive inside a footer file to be included. SSI directives can be contained in the included file, and includes can be nested - that is, the included file can include another file, and so on.

What else can I config?

In addition to being able to `config` the time format, you can also `config` two other things.

Usually, when something goes wrong with your SSI directive, you get the message

```
[an error occurred while processing this directive]
```

If you want to change that message to something else, you can do so with the `errmsg` attribute to the `config` function:

```
<!--#config errmsg="[It appears that you don't know how
to use SSI]" -->
```

Hopefully, end users will never see this message, because you will have resolved all the problems with your SSI directives before your site goes live. (Right?)

And you can `config` the format in which file sizes are returned with the `sizefmt` attribute. You can specify `bytes` for a full count in bytes, or `abbrev` for an abbreviated number in Kb or Mb, as appropriate.

Executing commands

Here's something else that you can do with the `exec` function. You can actually have SSI execute a command using the shell (`/bin/sh`, to be precise - or the DOS shell, if you're on Win32). The following, for example, will give you a directory listing.

```
<pre>
<!--#exec cmd="ls" -->
</pre>
```

or, on Windows

```
<pre>
<!--#exec cmd="dir" -->
</pre>
```

You might notice some strange formatting with this directive on Windows, because the output from `dir` contains the string ``<dir>" in it, which confuses browsers.

Note that this feature is exceedingly dangerous, as it will execute whatever code happens to be embedded in the `exec` tag. If you have any situation where users can edit content on your web pages, such as with a ``guestbook'', for example, make sure that you have this feature disabled. You can allow SSI, but not the `exec` feature, with the `IncludesNOEXEC` argument to the `Options` directive.

Advanced SSI techniques

In addition to spitting out content, Apache SSI gives you the option of setting variables, and using those variables in comparisons and conditionals.

Setting variables

Using the `set` directive, you can set variables for later use. We'll need this later in the discussion, so we'll talk about it here. The syntax of this is as follows:

```
<!--#set var="name" value="Rich" -->
```

In addition to merely setting values literally like that, you can use any other variable, including environment variables or the variables discussed above (like `LAST_MODIFIED`, for example) to give values to your variables. You will specify that something is a variable, rather than a literal string, by using the dollar sign ($) before the name of the variable.

```
<!--#set var="modified" value="$LAST_MODIFIED" -->
```

To put a literal dollar sign into the value of your variable, you need to escape the dollar sign with a backslash.

```
<!--#set var="cost" value="\$100" -->
```

Finally, if you want to put a variable in the midst of a longer string, and there's a chance that the name of the variable will run up against some other characters, and thus be confused with those characters, you can place the name of the variable in braces, to remove this confusion. (It's hard to come up with a really good example of this, but hopefully you'll get the point.)

```
<!--#set var="date" value="${DATE_LOCAL}_${DATE_GMT}" -->
```

Conditional expressions

Now that we have variables, and are able to set and compare their values, we can use them to express conditionals. This lets SSI be a tiny programming language of sorts. `mod_include` provides an `if`, `elif`, `else`, `endif` structure for building conditional statements. This allows you to effectively generate multiple logical pages out of one actual page.

The structure of this conditional construct is:

```
<!--#if expr="test_condition" -->
<!--#elif expr="test_condition" -->
<!--#else -->
<!--#endif -->
```

A *test_condition* can be any sort of logical comparison - either comparing values to one another, or testing the ``truth'' of a particular value. (A given string is true if it is nonempty.) For a full list of the comparison operators available to you, see the `mod_include` documentation.

For example, if you wish to customize the text on your web page based on the time of day, you could use the following recipe, placed in the HTML page:

```
Good <!--#if expr="%{TIME_HOUR} <12" -->
morning!
<!--#else -->
afternoon!
<!--#endif -->
```

Any other variable (either ones that you define, or normal environment variables) can be used in conditional statements. See Expressions in Apache HTTP Server for more information on the expression evaluation engine.

With Apache's ability to set environment variables with the `SetEnvIf` directives, and other related directives, this functionality can let you do a wide variety of dynamic content on the server side without resorting a full web application.

Conclusion

SSI is certainly not a replacement for CGI, or other technologies used for generating dynamic web pages. But it is a great way to add small amounts of dynamic content to pages, without doing a lot of extra work.