

## 1. What is an SSI?

- Server Side Includes are ‘commands’ for the web server
- SSIs are executed/evaluated when the page is delivered
- They are faster, simpler and more secure than CGI scripts
- They are not browser dependent: all the work is done at the server end
- Normally associated with a particular file suffix, eg .shtml

## 2. When should you use SSI?

- To include common elements in a pages (eg. a button bar)
- To include changing information in a static page (eg. todays’ headline)
- To offer different information based on the users identity (eg. hide internal links from the public)
- To improve web site management

Note that SSIs are dependent on support being made available in the server, and that there is a slight processing overhead.

## 3. Types of SSI

- Environment-specific information

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

- IF/ELSE instructions
- Text file inclusion

```
<!--#include file="text.txt"-->
```

## 4. Example of SSI

The SSI commands are included in the HTML file as comments; each command consists of a keyword (preceding by a #), and usually an attribute followed by an = sign, and a value in quotes:

```
<head>
<title>Example of SSI</title>
</head>
<!--#include file="layout_top.html"-->
<h1> Example of SSI </h1>
The layout of this page is fully controlled by a common file. There are
no BODY, COLOR or TABLE tags in this page.
<!--#include file="layout_bottom.html"-->
```

```
</html>
```

where `layout_top.html` has

```
<body bgcolor="red">
<table>
  <tr>
    <td valign="top"> side bar</td>
    <td valign="top">
```

and `layout_bottom.html` has

```
</td>
</tr>
</table>
```

## 5. SSI example of conditional text

This example shows different bits of text, depending on whether the reader's computer address is inside OUCS:

```
<!--#if expr="$REMOTE_ADDR = /^163.1.15./" -->
  This bit is only visible inside OUCS
<!--#else -->
  This bit is visible to everyone else
<!--#endif -->
```

## 6. SSI Elements

The allowed elements are:

- config** This command controls various aspects of the parsing. The valid attributes are:
  - errmsg** The value is a message that is sent back to the client if an error occurs whilst parsing the document.
  - sizefmt** The value sets the format to be used which displaying the size of a file. Valid values are `bytes` for a count in bytes, or `abbrev` for a count in Kb or Mb as appropriate.
  - timefmt** The value is a string to be used by the `strftime(3)` library routine when printing dates.
- echo** This command prints one of the include variables, defined below. If the variable is unset, it is printed as `(none)`. Any dates printed are subject to the currently configured `timefmt`. Attributes:
  - var** The value is the name of the variable to print.
- exec** The `exec` command executes a given shell command or CGI script. The valid attributes are:
  - cgi** The value specifies a (%-encoded) URL relative path to the CGI script. If the path does not begin with a `(/)`, then it is taken to be relative to the current document. The document referenced by this path is invoked as a CGI script, even if the server would not normally recognize it as such. However, the directory containing the script must be enabled for CGI scripts.
 

The CGI script is given the `PATH_INFO` and query string (`QUERY_STRING`) of the original request from the client; these cannot be specified in the URL path. The include variables will be available to the script in addition to the standard CGI environment.

If the script returns a `Location:` header instead of output, then this will be translated into an HTML anchor.

The `include virtual` element should be used in preference to `exec cgi`.

## Server-side includes

- cmd** The server will execute the given string using `/bin/sh`. The include variables are available to the command.
- fsize** This command prints the size of the specified file, subject to the `sizefmt` format specification. Attributes:
- file** The value is a path relative to the directory containing the current document being parsed.
  - virtual** The value is a (%-encoded) URL-path relative to the current document being parsed. If it does not begin with a slash (/) then it is taken to be relative to the current document.
- lastmod** This command prints the last modification date of the specified file, subject to the `timefmt` format specification. The attributes are the same as for the `fsize` command.
- include** This command inserts the text of another document or file into the parsed file. Any included file is subject to the usual access control. CGI scripts are invoked as normal using the complete URL given in the command, including any query string. An attribute defines the location of the document; the inclusion is done for each attribute given to the include command. The valid attributes are:
- file** The value is a path relative to the directory containing the current document being parsed. It cannot contain `../`, nor can it be an absolute path. The `virtual` attribute should always be used in preference to this one.
  - virtual** The value is a (%-encoded) URL relative to the current document being parsed. The URL cannot contain a scheme or hostname, only a path and an optional query string. If it does not begin with a slash (/) then it is taken to be relative to the current document. A URL is constructed from the attribute, and the output the server would return if the URL were accessed by the client is included in the parsed output. Thus included files can be nested.
- printenv** This prints out a listing of all existing variables and their values. No attributes.
- set** This sets the value of a variable. Attributes:
- var** The name of the variable to set.
  - value** The value to give a variable.

For example: `<!--#set var="category" value="help" -->`

## 7. Standard information available

You can access all the standard CGI variables, listed below. This means that you can say, eg, `<!--#echo var="REMOTE_HOST" -->` to print the name of the computer which has asked for the page.

- SERVER\_SOFTWARE** The name and version of the information server software answering the request (and running the gateway). Format: name/version
- SERVER\_NAME** The server's hostname, DNS alias, or IP address as it would appear in self-referencing URLs.
- GATEWAY\_INTERFACE** The revision of the CGI specification to which this server complies. Format: CGI/revision
- SERVER\_PROTOCOL** The name and revision of the information protocol this request came in with. Format: protocol/revision
- SERVER\_PORT** The port number to which the request was sent.
- REQUEST\_METHOD** The method with which the request was made. For HTTP, this is "GET", "HEAD", "POST", etc.
- PATH\_INFO** The extra path information, as given by the client. In other words, scripts can be accessed by their virtual pathname, followed by extra information at the end of this path. The extra information is sent as `PATH_INFO`. This information should be decoded by the server if it comes from a URL before it is passed to the CGI script.
- PATH\_TRANSLATED** The server provides a translated version of `PATH_INFO`, which takes the path and does any virtual-to-physical mapping to it.
- SCRIPT\_NAME** A virtual path to the script being executed, used for self-referencing URLs.

**QUERY\_STRING** The information which follows the ? in the URL which referenced this script. This is the query information. It should not be decoded in any fashion. This variable should always be set when there is query information, regardless of command line decoding.

**REMOTE\_HOST** The hostname making the request. If the server does not have this information, it should set **REMOTE\_ADDR** and leave this unset.

**REMOTE\_ADDR** The IP address of the remote host making the request.

**AUTH\_TYPE** If the server supports user authentication, and the script is protected, this is the protocol-specific authentication method used to validate the user.

**REMOTE\_USER** If the server supports user authentication, and the script is protected, this is the username they have authenticated as.

**REMOTE\_IDENT** If the HTTP server supports RFC 931 identification, then this variable will be set to the remote user name retrieved from the server. Usage of this variable should be limited to logging only.

**CONTENT\_TYPE** For queries which have attached information, such as HTTP POST and PUT, this is the content type of the data.

**CONTENT\_LENGTH** The length of the said content as given by the client.

## 8. Other Include Variables

In addition to the variables in the standard CGI environment, these are available for the `echo` command, for `if` and `elif`, and to any program invoked by the document.

**DATE\_GMT** The current date in Greenwich Mean Time.

**DATE\_LOCAL** The current date in the local time zone.

**DOCUMENT\_NAME** The filename (excluding directories) of the document requested by the user.

**DOCUMENT\_URI** The (%-decoded) URL path of the document requested by the user. Note that in the case of nested include files, this is *not* then URL for the current document.

**LAST\_MODIFIED** The last modification date of the document requested by the user.

## 9. Flow Control Elements

These are available in Apache 1.2 and above. The basic flow control elements are:

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

The `if` element works like an `if` statement in a programming language. The test condition is evaluated and if the result is true, then the text until the next `elif`, `else`, or `endif` element is included in the output stream.

The `elif` or `else` statements are used to put text into the output stream if the original `test_condition` was false. These elements are optional.

The `endif` element ends the `if` element and is required.

*test\_condition* is one of the following:

*string* true if *string* is not empty

*string1* = *string2* *string1* != *string2* Compare *string1* with *string2*. If *string2* has the form */string/* than it is compared as a regular expression. Regular expressions have the same syntax as those found in the Unix `egrep` command.

( *test\_condition* ) true if *test\_condition* is true

! *test\_condition* true if *test\_condition* is false

*test\_condition1* && *test\_condition2* true if both *test\_condition1* and *test\_condition2* are true

*test\_condition1* || *test\_condition2* true if either *test\_condition1* or *test\_condition2* is true

"=" and "!=" bind more tightly than "&&" and "||". "!" binds most tightly. Thus, the following are equivalent:

```
<!--#if expr="$a = test1 && $b = test2" -->
<!--#if expr="($a = test1) && ($b = test2)" -->
```