SSI+ 1.0 Reference Manual

SSI+ 1.0

(*The SSI+ specification is in development. Comments, suggestions, and questions should be directed to the author: Mark West, Questar Microsystems Inc. at markw@mailbox.theworld.com).*
The SSI+ 1.0 specification is implemented on WebQuest and WebQuest95.
For information on WebQuest and WebQuest see: http://www.questar.com

Server side includes (SSI)s applied to an HTML document, provide for interactive real-time features such as echoing current time, conditional execution based on logical comparisons, querying or updating a database, sending an **email**, etc., with no programming or CGI scripts. An SSI consists of a special sequence of characters(tokens) on an HTML page. As the page is sent from the HTTP server to the requesting client, the page is scanned by the server for these special tokens. When a token is found the server interprets the data in the token and performs an action based on the token data.
The format of a SSI token is as follows : '<!--#'<tag><variable set> '-->'
where :

- '<--#' is the opening identifier, a SSI token always starts with this.
- <tag> is one of the following: echo, include, fsize, flastmod, exec, config, odbc, email, if, goto, label, break
- <variable set> is a set of one or more variables and their values. The values allowed here and dependent on the <tag>and are listed below each tag listed below. The format of a variable set is as follows : <variable name> '=' '"' variable data '"' <variable name2> '=' '"' variable data2 '"' <variable name n> '=' '"' variable datan '"'
- "-->" is the closing identifier, a SSI token always ends with this.

SSI tokens may contain special tags (subtokens) which are dereferenced before evaluation of the SSI token takes place. Subtokens maybe inserted any place in the SSI token. Subtokens are especially useful when forming if, odbc, email, and exec tokens (described below) that are based on HTML form data returned from a remote client. The format of a subtoken is as follows: '&&'<subtokendata>'&&' where:

- '&&' is a reserved character sequence defining the beginning and end of the subtoken
- <subtokendata> is any value allowed in an echo token.

SSI Documents

An SSI+ document must have a file extension of '.SHT' or '.SHTM'. For the sake of efficiency the HTTP server will only scan those documents with the aforementioned extensions for SSI+ tokens.

SSI Tags, Variables and Data

The following is a list of the cuttently supported SSI+ tags:

Echo tag provides for inserting the data of certain variables into an HTML page.

Include tag provides for inserting the contents of a file into the HTML page at the location of the include token.

Fsize tag provides for inserting the size of a given file into the HTML page at the location of the fsize token.

Flastmod tag provides for inserting the last modification date of a given file into the HTML page at the location of the flastmod token.

Exec tag provides for executing an external executable.

Config tag provides for setting certain HTML output options.

Odbc tag provides for querying and updating ODBC databases.

Email tag provides for sending an email whenever an HTML page is accessed or an HTML form is submitted.

If tag provides for conditional execution of SSI operations and conditional printing of HTML text based on logical comparisons.

Goto tag provides for jumping to a label token without executing any SSI code or printing any HTML text between the goto token and label token.

Label tag provides a place for a goto or if goto token to jump to.

Break tag provides for termination of HTML documents at any point.

Echo Tag

The **echo** tag provides for inserting the data of certain variables into an HTML page. The only variable under the **echo** tag is '**var**'. The data in an echo token is translated into a string that depends on the value in the variable and that string is inserted into the HTML page at the location of the echo token in the HTML page.

Example. The following string in an HTML page :

```
The Greenwich Mean Time is <!--#echo var="DATE_GMT"--> and I am so happy to be here.
```

would resolve at runtime into something like:

```
The Greenwich Mean Time is Fri Jul 21 21:24:48 1995 and I am so happy to be here.
```

The values allowed as variable data are available form two sources: Form fields and environment variables.

Form Fields are those datum which are available when a POST operation is performed on a SSI enabled HTML document from an HTML form. Each form field may be referenced by name.

Example. Suppose one has an HTML form with several fields defined, one of which is named 'First Name'. When an HTTP client POSTs the form to an SSI enabled HTML document with an echo token: <--#echo var="First Name" -->, the HTML document will then have the contents of the supplied 'First Name' field inserted upon transmission to the client.

Environment variables are a set of datum defined by the local server and remote client and are defined below.

DOCUMENT_NAME" This variable is the complete local directory path of the current document.

DOCUMENT_URI" This variable is the local path of the current document referenced to the base directory of the webspace.

QUERY_STRING_UNESCAPED" This variable is the unescaped query string sent by the client browser, all shell-special characters escaped with \.

DATE_LOCAL" This variable is current local date and time.

DATE_GMT This variable is the current Greenwich Mean date and time.

LAST_MODIFIED This variable is the date and time of the last modification of the current document.

REMOTE_ADDR This variable is the IP address of the remote client browser.

QUERY_STRING This variable is the raw query string sent from the remote browser.

SERVER_SOFTWARE This variable is the name of the HTTP server software.

SERVER_NAME This variable is the local computer name of the HTTP server.

GATEWAY_INTERFACE This variable is the name/version of the Common Gateway Interface served on this HTTP server.

SERVER_PROTOCOL This variable is the name/version of HTTP served on this HTTP server

SERVER_PORT This variable is the IP port the HTTP server is answering on.

REQUEST_METHOD This variable is the method by which the current document was requested.

PATH_INFO This variable is the extra path info that is sent. This information is regarded as virtual (the path is relative to the base directory of the HTTP server).

PATH_TRANSLATED This variable is the 'PATH_INFO' variable translated from virtual to local (physical) disk location.

SCRIPT_NAME This variable is the virtual path of the script being executed.

REMOTE_HOST This variable is the host name of the remote client.

AUTH_TYPE This variable is the authentication method used to validate the remote client.

REMOTE_USER This variable is the user name used to validate authentication from the remote client.

REMOTE_IDENT This variable is the remote user name if supporting RFC931 identification.

CONTENT_TYPE This variable is the content type of the attached information in the case of a POST or PUT.

CONTENT_LENGTH This variable is the length of the attached information in the case of a POST or PUT.

HTTP_ACCEPT This variable is a comma separated list of mime types that are accepted by the remote browser.

HTTP_USER_AGENT This variable is the name of the remote client browser software.

REFERER This variable is the ULR of the HTML document which referred the remote client to this document.

FROM This variable is the name (most likely the-mail address) of the remote client user.

FORWARDED This variable is the name of the proxy server through which this document is being processed.

ACCEPT_LANGUGE This variable lists the human languages that are acceptable to the remote client.

HTTP_COOKIE This variable contains the cookie sent by the remote client, and is explained in detail below.

Include Tag

The **include** tag provides for inserting the contents of a file into the HTML page at the location of the include token. The SSI+ include tag is fully recursive, each document inserted may itself contain further SSI+ insert tokens or any other SSI+ tokens.

virtual The virtual variable is used to specify a file path/name relative to the base directory of the HTTP server. The format is: 'virtual="'<filename>'"' where <filename>a file path/name relative to the base directory of the HTTP server.

file The file variable is used to specify a file path/name relative to the directory of the current document. The format is: 'file="'<filename>'"' where <filename>a file path/name relative to the directory of the current document.

Example. The following token on an HTML document inserts all text and SSI+ tokens from the file <html base directory>'SSI\INSERT.SHT'into the current document before transmission back to the client browser:

```
<!--#include virtual="SSI\INSERT.SHT" -->
```

FSize Tag

The **fsize** tag provides for inserting the size of the given file into the HTML page at the location of the fsize token.

virtual The virtual variable is used to specify a file path/name relative to the base directory of the HTTP server. The format is: 'virtual="'<filename>'"' where <filename>a file path/name relative to the base directory of the HTTP server.

file The file variable is used to specify a file path/name relative to the directory of the current document. The format is: 'file="'<filename>'"' where <filename>a file path/name relative to the directory of the current document.

Example. The following token on an HTML document inserts the size of file <html base directory>'SSI\INSERT.SHT' into the current document before transmission back to the client browser:

```
<!--#fsize virtual="SSI\INSERT.SHT" -->
```

FLastMod Tag

The **flastmod** tag provides for inserting the size of the given file into the HTML page at the location of the flastmod token.

virtual The virtual variable is used to specify a file path/name relative to the base directory of the HTTP server. The format is: 'virtual="'<filename>'"' where <filename>a file path/name relative to the base directory of the HTTP server.

file The file variable is used to specify a file path/name relative to the directory of the current document. The format is: 'file="'<filename>'"' where <filename>a file path/name relative to the directory of the current document.

Example. The following token on an HTML document inserts the size of file <html base directory>'SSI\INSERT.SHT' into the current document before transmission back to the client browser:

```
<!--#fsize virtual="SSI\INSERT.SHT" -->
```

Exec Tag

The **exec** tag provides for executing an external executable system command.

cmd The cmd variable is used to specify the name and command line parameters of any shell executable command.

The format is: 'cmd="'<exename> <argument list>'"' where<exename> is the path and/or name of the executable command and <argument list> is the list of command line arguments to send to the executable command. If a full path is not specified then the 'PATH' environment variable of the server will be searched for the executable. A shell executable is any executable that may be executed on the console, this allows web administrators and authors to use executables that accept command line arguments as an alternative to CGI executables (that accept only environment variables). The output of shell executables may be echoed into the HTML document, see the **'config..cmdecho'** tag for details.

cgi The cgi variable is used to specify the name a CGI executable(script) relative to the base directory of the HTTP server.

The format is: 'cgi="'<exename>"' where <exename>is the path and name of the CGI executable relative to the base directory of the web space. The CGI will be executed and any cgi output will be inserted into the current HTML document at the location of the cgi token.

Config Tag

The **config** tag provides for setting certain HTML output options.

errmsg The errmsg variable is used to set the error message that gets printed when the SSI+ engine encounters a parsing error or unavailable required data. This variable is retained for compatability with standard SSI, you may wish to use the **onerr** variable instead.

timefmt The timefmt variable is used to set the format of echo..time SSI+ token output.

sizefmt The sizefmt variable is used to set the format of echo..size SSI+ token output.

cmdecho The cmdecho variable is used to set the output option of subsequent **exec..cmd** tokens.

The format is 'cmdecho'='"<onoroff>"'where <onoroff> is either '**ON**' or '**OFF**'. When the SSI+ parsing engine encounters an **exec..cmd** token it executes the command. If the command returns output then that output may be echoed into the HTML document or it may be ignored. The format of the data echoed is dependent on the presence or absence of **config..cmdprefix** and **config..cmdpostfix** tokens in the document. In the absence of **config..cmdprefix** and **config..cmdpostfix** tokens the output will be echoed exactly as returned with no formatting and no special character interpretation. In the presence of **config..cmdprefix** and/or **config..cmdpostfix** tokens the output will be formatted and interpreted. To activate echoing set cmdecho to '**ON**' otherwise set it to '**OFF**'. The default is '**OFF**'.

cmdprefix The cmdprefix variable is used to set the string prefixed to each line out output from subsequent **exec..cmd** tokens.

The format is 'cmdprefix="'<string>"'where <string> is any character string and/or HTML format tags. When the SSI+ parsing engine encounters an **exec..cmd**token it executes the command. If the command returns output then that output may be echoed into the HTML document or it maybe ignored. If the output is echoed (see **'cmdecho'** above),then each line output from the executable will be prefixed with the string supplied before being echoed into the HTML document.

cmdpostfix The cmdpostfix variable is used to set the string appended to the end of each line out output from subsequent **exec..cmd** tokens.

The format is 'cmdpostfix="'<string>"'where <string> is any character string and/or HTML format tags. When the SSI+ parsing engine encounters an **exec..cmd**token it executes the command. If the command returns output then that output may be echoed into the HTML document or it maybe ignored. If the output is echoed (see **'cmdecho'** above),then each line output from the executable will be appended with the string supplied before being echoed into the HTML document.

onerr The onerr variable is used to set the action to be taken when the SSI+ engine encounters an error. The format is 'onerr="'<action>"'where <action> is one of the following tags.

'goto ' causes a jump to a label token (see below).

The format of the goto tag is:

'goto ' <label >

where <label > is the name of a label defined in a subsequent label tag (see below).

'print' causes text to be printed.

The format of the print tag is:

'print "'<text>"'

where <text> is any HTML text or tag.

'error' causes the current **config..error** message to be printed.

'break ' causes termination of the HTML document transmission to the client.

'errorbreak' causes the current **config..error** message to be printed, and then causes termination of the HTML document transmission to the client.

'printbreak' causes text to be printed, and then causes termination of the HTML document transmission to the client. The format of the printbreak tag is the same as the format of the print tag.

Example. The following token on an HTML document sets the SSI+ error message to '*** ERROR ***'. From this point on down when an error occurs in the SSI+ parsing engine the message '*** ERROR ***'will be inserted into the HTML document at the location of the offending SSI+ statement.

```
<!--#config errmsg="*** ERROR ***" -->
```

Example. The following token on an HTML document sets the SSI+ error action to print a message and terminate the document. From this point on down when an error occurs in the SSI+ parsing engine the message will be inserted into the HTML document at the location of the offending SSI+ statement, and the document will be terminated

```
<!--#config onerr="printbreak "Sorry, we encountered an error while
processing your document."" -->
```

Example. Suppose you wish to create an HTML document that performs a 'PING' operation on address '204.96.64.171' and then echo the results back to the client browser, with each line echoed as an element in an unnumbered list.

Insert the following lines into your HTML document:

```
<UL>
<!--#config cmdecho="ON" -->
<!--#config cmdprefix="<LI>" -->
<!--#exec cmd="ping 204.96.64.171 -w 20000" -->
</UL>
```

When the document is accessed by a remote browser the output would look something like this:

```
Pinging 204.96.64.171 with 32 bytes of data:
Reply from 204.96.64.171: bytes=32 time<10ms TTL=32
Reply from 204.96.64.171: bytes=32 time<10ms TTL=32
Reply from 204.96.64.171: bytes=32 time<10ms TTL=32
Reply from 204.96.64.171: bytes=32 time<10ms TTL=32
```

ODBC Tag

The **odbc** tag provides for querying and updating odbc databases. Four variables are defined for the odbc token; 'debug', 'connect', 'statement', and 'format'.

DEBUG
The **debug** variable is used to set the SSI+ engine into debug mode. Debug mode provides diagnostics of a highly detailed nature from both the SSI engine itself and the local ODBC engine. Debug messages appear on the returned HTML document at the position at which the errors occur. The debug variable should be used only for development and must be removed before production. When the debug variable is present a warning message will appear indicating it's presence, this message is benign and will not affect any other aspect of the SSI+ engine.

The format of the debug variable is : 'debug='"<debugstring>"' where;

<debugstring> is any string and is reserved for future use.

CONNECT
The **connect** variable is used to connect to a pre-existing odbc data source, to allow for subsequent **statement** tag operations on that data source.

The format of the connect variable is
'connect="'<datasource>','<username>','<password>'"'where ;

<datasource> is the name odbc data source as defined on the local system in the odbc configuration utility. **CAUTION!** the account under which the server is run must be granted permission to access the data source.

<user name> is the name which to log into the data source.

<password> is the password with which to access the data source.

Example. To connect to a data source called 'odbcsht' as user 'dufus' and password 'dorkboy', one would use the following statement: <!--#odbc connect="odbcsht,dufus,dorkboy"-->.

STATEMENT
The **statement** variable is used to submit a Transact SQL statement to the odbc data source. The format of a statement variable is as follows:
'statement="'<SQLStatement>'"',where:

<SQL Statement> is any Transact SQL statement as defined in odbc and SQL reference text and help files.

Example. Suppose one wanted to query the 'CUSTOMERS' table from the above connected 'odbcsht' database to return all rows and display each row on a separate line. One may use the following sequence of statements:

- Connect to the database with a **connect** token as described above.
- Setup the output format with a **statement** token as described below.
- Execute the query: <!--#odbc statement="SELECT NAME, AGE, VISCOSITY FROM CUSTOMERS ORDER BY 3, 2, 1" -->
- Each row of the database will the be inserted into the HTML page per the format statement as demonstrated below.

FORMAT
The **format** variable is used to provide a template for the format of data that is returned from and odbc query. Use this variable to set up the appearance of data that will be returned from subsequent **statement** tag operations that return data from a database

(i.e. the SQL statement 'SELECT').

The format of the format variable is 'format="'<cprintfstatement>'"'where;

<cprintfstatement> is a standard C language printf format string with the restriction of only allowing string (%s) insertions. The user is referred to any C language text for a description of this format. The number of instances of %s must be equal to the number of fields selected in a the subsequent SQL SELECT statement token.

Example. Suppose one wanted to query the 'CUSTOMERS' table from the above connected 'odbcsht' database to display the columns 'name', 'age', and 'viscosity' with each row on a separate line. One may use the following sequence of statements:

1. Connect to the database with a **connect** token as described above.
2. Setup the output format: <!--#obdc format="<P>Thecustomer's name is %s, and he is %s years old, he prefers a motor oil with SPF %s viscosity" -->.
3. Execute the query with a **statement** token as described above.
4. Each row of the database will the be inserted into the HTML page per the format statement. For example if the database has 3 rows the HTML output would look something like this:

```
Customer's name is Conan, and he is 29 years old, he prefers a motor oil
with SPF 15 viscosity
Customer's name is Kevin, and he is 45 years old, he prefers a motor oil
with SPF 30 viscosity
Customer's name is Alan, and he is 43 years old, he prefers a motor oil
with SPF 50 viscosity
```

EMail Tag

The **email** tag provides for sending an Email whenever an HTML page is accessed or an HTML form submitted. The nature of the variables below is defined in RFC 733. The variables 'fromhost', 'tohost', 'fromaddress' and 'toaddress' are required, all others are optional.

fromhost defines the name of the smtp host sending the mail.

tohost defines the name of the smtp host the mail will be sent to.

fromaddress defines the email address from party.

toaddress defines the email address of the recipient party.

message defines the message body to be sent.

subject defines the subject field of the message to be sent.

sender defines the email address sending party.

replyto defines the email address to which replies should be sent.

cc defines the courtesy copy email addresses.

inreplyto defines the inreplyto field of the message to be sent.

id defines the id field of the message to be sent.

Example. The following document send an email. Suppose we have a form with datum : [First, Last, Middle Initial, Company, Address1, Address2, City, State, Zip, Country, Phone, Fax, Request, Urgency, ReplyMethod; Email, Subject, Message] we may post that form to an HTML document containing the following fragment to send an email.

```
<!--#email fromhost=""www.theworld.com"tohost="mailbox.theworld.com"
message="First -&&First&&, Last - &&Last&&, MI- &&Middle Initial&&, Company
- &&Company&&, Address - &&Address1&&, &&Address2&&, &&City&&, &&State&&,
&&Zip&&, &&Country&&, Phone - &&Phone&&, Fax - &&Fax&&, Request &&Urgency
&&via &&ReplyMethod&&, Message - &&Message&& "fromaddress="&&EMail&&"
toaddress="markw@mailbox.theworld.com"subject="WebMan - &&Subject&&"
sender="&&EMail&& "replyto="&&EMail&&" cc=" "inreplyto="A WebMan Automated
E-Mail" id="WebManEMail" -->
```

If Tag

The **if** tag provides for conditional execution of SSI operations, and conditional printing of HTML text, based on logical comparisons. The format of the if tag is :
'if' '"'<operand1>'"' <operator> '"'<operand2>'"'<operation>
where:

<operand1> is the first operand of a logical comparison statement

<operand2> is the second operand of a logical comparison statement

<operator> is the logical comparison method ['==', '!=','<', '>', '!<', '!>']

<operation> is the action to take if the logical comparison evaluates to TRUE ['goto ', 'print', 'error', 'break ', 'errorbreak', 'printbreak']

The operands may be any string or number (integer or floating point). In the event that both operands are numbers the comparison will be based on the value of the numbers. In the event that one or both of the operands and not numbers, the comparison will be based on the alphabetic order of the operands.
The special case of the NULL operand is defined by two quotes with no characters between them. The NULL operand may used to check for the existence of form data from the remote client (see example below).

The operator defines what kind of comparison is performed on the operands:

== The equalto operator evaluates to TRUE if the operands are equal to each other.

!= The notequalto operator evaluates to TRUE if the operands are not equal to each other.

< The lessthan operator evaluates to TRUE if operand1 is less than operand2.

> The greaterthan operator evaluates to TRUE if operand1 is greater than operand2

!< The notlessthan operator evaluates to TRUE if operand1 is not less than operand2

!> The notgreaterthan operator evaluates to TRUE if operand1 is not greater than operand2

hasstring The hasstring operator returns TRUE is the text string in operand2 is found in the operand1 string.

In the event that the logical comparison evaluates to FALSE, nothing happens, If the logical comparison evaluates to TRUE then one of the following operations may be performed:

'goto ' causes a jump to a label token (see below).

The format of the goto tag is:
'goto ' <label >
where <label > is the name of a label defined in a subsequent label tag (see below).

'print' causes text to be printed.

The format of the print tag is:
'print' <text>
where <text> is any HTML text or tag.

'error' causes the current **config..error** message to be printed.

'break ' causes termination of the HTML document transmission to the client.

'errorbreak' causes the current **config..error** message to be printed, and then causes termination of the HTML document transmission to the client.

'printbreak' causes text to be printed, and then causes termination of the HTML document transmission to the client. The format of the printbreak tag is the same as the format of the print tag.

Example. The following document fragment compares two numbers, if the operands are not equal then a goto will jump to a label.

```
<!--#if "10" != "20" goto test label -->
<P>This should not print
<!--#label ="testlabel " -->
<P>This should print
```

Example. The following document fragment demonstrates conditional execution based on data delivered from an HTML form. Suppose we have two form datum called 'formdata1' and 'formdata2' and we wish to compare them. The following document fragment compares the two operands, if the operands are equal then a goto will jump to a label. Otherwise the next line will print and the document will terminate on a break token (see below).

```
<!--#if "&&formdata1&&" == "&&formdata2&&" goto testlabel -->
<P>The operands are not equal.
<!--#break -->
<!--#label ="testlabel " -->
<P>operands are equal.
```

Example. The following document fragment prints two different statements depending on whether or not the client agent is NCSA Mosaic.

```
<!--#if "&&HTTP_USER_AGENT&&" hasstring "Mosaic" goto mosaiclabel -->
<P>You are not using Mosaic
<!--#goto ="defaultlabel" -->
<!--#label ="mosaiclabel" -->
<P>You are using Mosaic
<!--#label ="defaultlabel" -->
```

Example. Suppose we have a form with amongst other things a field named "BOO" and we wish to make sure that the remote client user enterted data into the "BOO" field before submitting the form. The following document fragment checks for the presence of data in the "BOO" field, if data exists then nothing happens, if data does not exist then a message will be displayed and the document will terminate.

```
<!--#if "BOO" == "" printbreak "<P>You must provide data for the BOO
field, please resubmit." -->
```

Goto Tag

The **goto** tag provides for jumping to a label token without executing any SSI code or printing any HTML text between the goto token and the label token. The format of the goto tag is:

'goto ="'<label >'"'

where <label > is the name of a label defined in a subsequent label tag (see below).

Example. The following document fragment demonstrates a goto to a label.

```
<!--#goto ="testlabel" -->
<P>This line will not print.
<!--#label ="testlabel" -->
<P>This line will print.
```

Label Tag

The **label** tag provides a place for a goto or if**..**goto token to jump.

The format of the label tag is:

'label ="'<label>'"'

where <label> is any string less than 51 characters long without space(' ') characters.

When the SSI+ engine encounters a label token nothing happens, it is simply a place holder for a previous goto to jump to.

Break Tag

The **break** tag provides for termination of the HTML document at any point. When the SSI+ engine encounters a break token, the HTML document is immediately truncated and transmission to the client is ended.

Example. The following document fragment demonstrates a break token.

```
<P>This line will print.
<!--#break -->
<P>This line will not print because the document has been truncated and
transmission to the client is terminated.
```