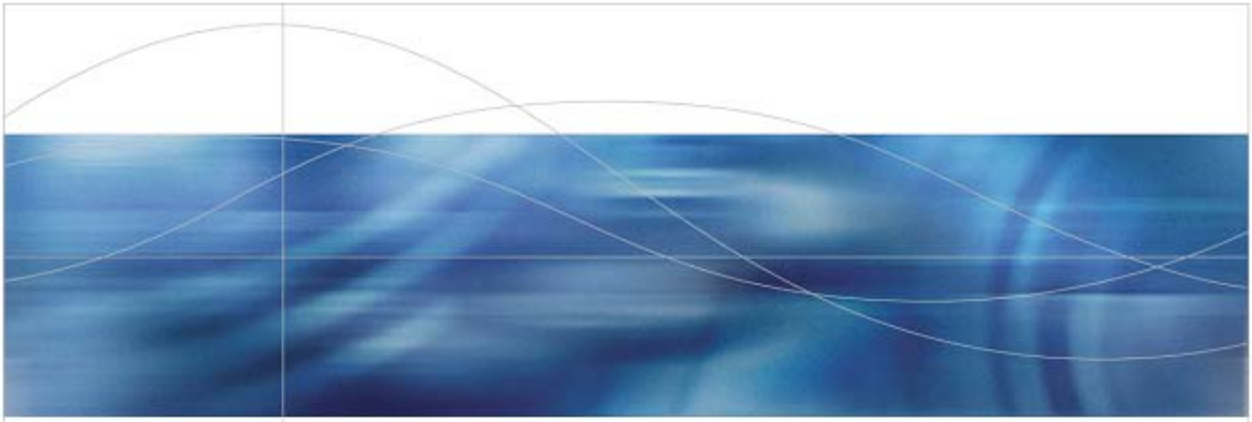
August 29, 2004



EdgeSuite 5.0: ESI Developer's Guide

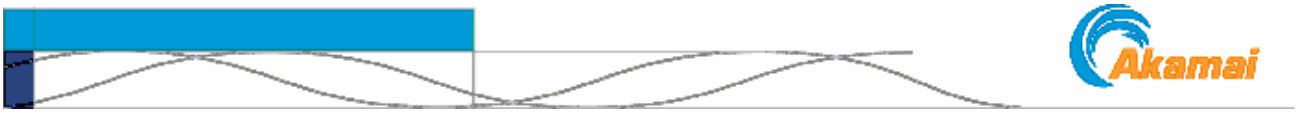
Using Edge Side Includes

Akamai

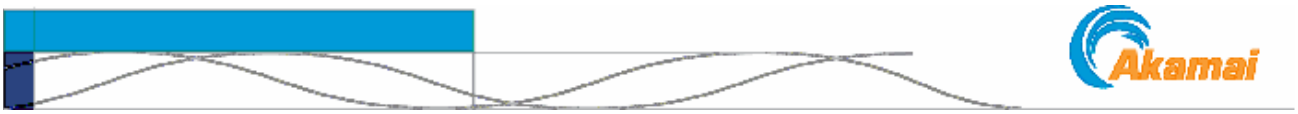
EdgeSuite 5.0: ESI Developer's Guide

Viewing this Document in Adobe Acrobat Reader

If you are reading this document on screen using Acrobat Reader, note that the document is fully hypertext enabled. That is, the table of contents entries, cross-references, and indices, if these exist, are linked to the pages they reference. When you see a page number in a table, index, or reference, you can click the number to jump directly to the referenced page.



CONTENTS



CHAPTER 1. About Edge Side Includes

The EdgeSuite Edge Side Includes (ESI) service provides for dynamically generating HTML pages at the edge of the Internet, near the end user.

The ESI language is an XML-based markup language that provides the tools to assemble the content dynamically on Akamai's network.

About this Guide

This guide is for developers using ESI to develop web content, and it covers the following topics:

- A summary of EdgeSuite ESI features and the relation of EdgeSuite ESI to the ESI 1.0 Specification
- How it works—an overview to building documents using the ESI language and how dynamic documents are delivered
- Before you begin: about setting EdgeSuite configuration and using control mechanisms such as HTTP headers
- Akamai resources related to ESI
- The ESI language—a quick reference to the elements, followed by a description of the elements, in the following order:
  - A quick reference to the language and its syntax
  - The **`include`** statement, the basic ESI function, and the eval statement, which provides for processing a child's variables in the parent's namespace.
  - Iteration and conditional inclusion
  - Alternative processing and exception handling
  - Expressions and operations
  - Environment and user-defined variables support
  - Functions
  - User-defined functions
  - Internationalization—using multibyte character sets
- Content control and Configuration
- Error and exception handling
- Examples

The ESI Specification

The EdgeSuite Edge Side Includes (ESI) service was formerly known as Akamai Side Includes (ASI). This name change reflects the service's adherence to the ESI 1.0 specification. The ESI 1.0 specification is an open specification co-authored by Akamai and 14 other industry leaders, the purpose being to develop a uniform programming model to provide the ability to build dynamic pages at the edge of the internet, close to the end user.

New in 5.0

The EdgeSuite 5.0 ESI implementation conforms to the ESI 1.0 specification, and also contains the following significant extensions:

- A new statement, **`esi:break`**, can be used to exit from an **`esi:foreach`** iteration. See page 35.
- On a Beta basis, ESI now provides for user-defined functions. See page 83.
- A new range operator can be used in expressions. See page 65.
- Optionally, you can enable the use of bitwise operators to act on the internal binary representation of a number. Enabling the operators sets && and || as logical operators and freeing & and | for use as bitwise operators. The bitwise behavior is off by default but can be enabled in your EdgeSuite configuration. See pages 60 and 64.
- New ESI functions, **`$base64_encode()`**and **`$base64_decode()`**, and **`$digest_md5()`** and **`$digest_md5_hex()`**, provide for Base64 encoding and decoding, and MD5 digests. See page 79.

Previous Versions

The previous versions of EdgeSuite ESI also contained significant extensions to the ESI 1.0 specification. These are indicated by footnotes in the text of this document.

Support for Prior Code

EdgeSuite 5.0 ESI supports code written for 4.0 – 4.8.x. Also, code written for ASI 1.2 and 1.3 is supported. EdgeSuite will read and process both **`<asi:…>`** and **`<esi:…>`** tags for v1.2 and v1.3 code.

Features

EdgeSuite shares the same reliability, fault-tolerance, performance, and scalability found in Akamai's FreeFlow technology. ESI can improve site performance by caching the objects that comprise dynamically generated HTML pages at the edge of the Internet, close to the end user. ESI allows for dynamic content assembly at the edge.

As the content provider, you design and develop the business logic to form and assemble the pages, using the ESI language within your content development format.

The ESI language, which finds its point of departure in Server Side Includes (SSI) implemented in Apache and other Web servers, includes the following features:

- **Inclusion**—the central ESI feature is the ability to fetch and include files to comprise a Web page, with each file subject to its own configuration and control—its own specified time-to-live in cache, revalidation instructions, and so forth. Included documents can include ESI markup for further ESI processing. Currently, ESI supports the inclusions nested up to fifteen levels.
- **Integration with Edge Transformations**—ESI can include fragments processed by the Edge Transformations Service, and beyond that, has the ability to create and pass variables and global params, specify an XSL stylesheet, and specify XSLT processing itself.
- **Environmental variables**—ESI supports the use of standard CGI environment variables such as cookie information and POST responses. These variables can be used inside ESI statements or outside of ESI blocks.
- **User-defined Variables**—ESI supports a range of user-defined variable types.
- **Functions, and the ability to create user-defined functions**—ESI supports functions to perform various evaluations, for example, to set HTTP headers, set redirects, and create time stamps.
- **Conditional logic**—ESI supports conditional logic based on Boolean expressions and environmental variables comparisons.
- **Iteration**—ESI provides a logic to iterate through lists or dictionaries.
- **Secure Processing**—ESI supports a logic for SSL processing, automatically using secure processing for fragments if the template is secure.
- **Exception and error handling—**ESI allows you to specify alternative objects and default behavior such as serving default HTML in the event that an origin site or document is not available. Further, it provides an explicit exception-handling statement. If a severe error is encountered while processing a document with ESI markup, the content returned to the end user can be specified in a "failure action" configuration option associated with the ESI document.

Content Control and Configuration

EdgeSuite and EdgeSuite ESI provide several different mechanisms for content control and the tuning of parameters. See "Configuration & Content Control" on page 89.

Overview—Building Documents with Dynamic Content

Template and Fragments

The basic structure you use to create dynamic content in ESI is a *template* page containing *HTML fragments*.

The template page consists of common elements such as logo, navigation bars, framework, and other "look and feel" elements of the page. The HTML fragments represent dynamic subsections of the page.

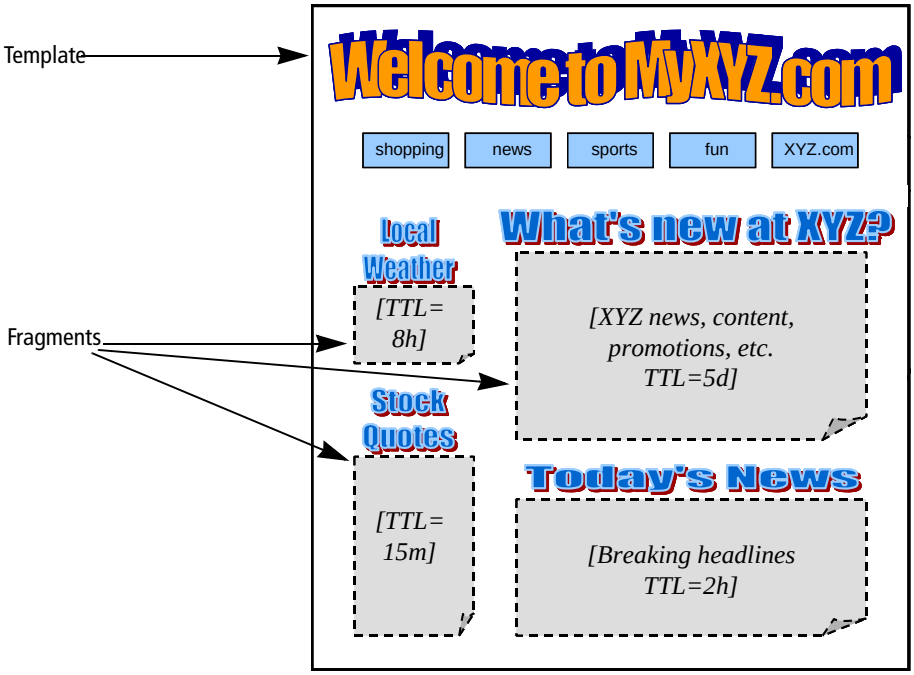


Figure 1. Template and HTML Fragments

The template is the file associated with the URL the end user requests. It is marked up with ESI language that tells EdgeSuite to fetch and include the HTML fragments. The fragments themselves are HTML- or XML-marked up files containing discrete text or other objects.

Each fragment is treated as its own separate object on the Akamai network—each with its own cacheability and access profiles set by way of headers or configuration files. You may want to cache the template for several days, but cache a particular fragment containing a story or ad for a matter of minutes or hours. You may want to set up particular fragments not to cache at all.

How ESI Delivers Dynamic Pages

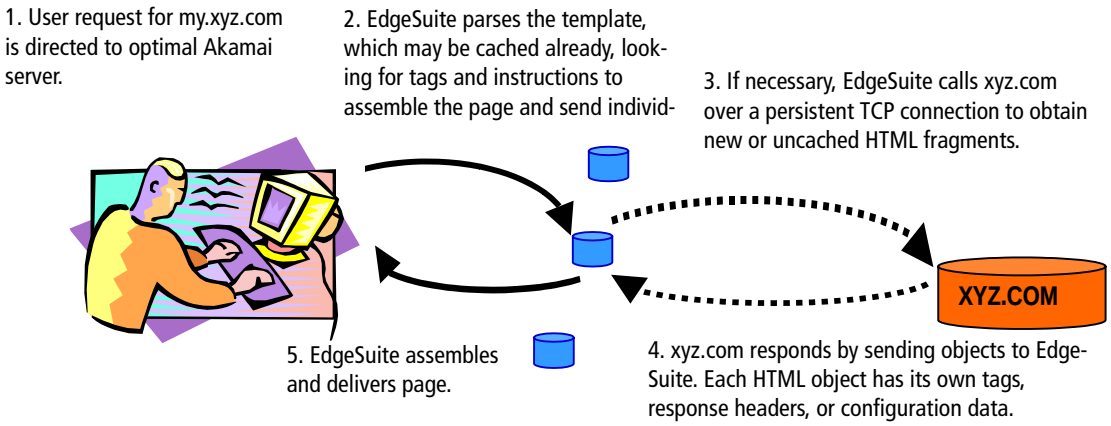


Figure 2. Edge Side Includes: How it Works

1. When the user requests the content page, EdgeSuite directs the request to the optimal (for the user) Akamai server.
2. The template page associated with the request may already be cached, since it may contain persistent, frequently used material. If the template isn't cached, EdgeSuite fetches it from xyz.com.
3. When EdgeSuite sees the ESI language markup in the template, it reads the tags and instructions, conditions, and variables.
4. EdgeSuite calls xyz.com to request or validate any fragments.
5. The origin, xyz.com, sends new objects back to EdgeSuite. Each object is an HTML fragment with its own associated configuration and header data, and it can include other fragments up to fifteen nested levels starting with the template.
6. EdgeSuite assembles and delivers the custom page to the user, and also caches appropriate objects for further use.

Additional Resources

Related Akamai resources include the following documents, available to customers and reseller on the Akamai customer portal, https://control.akamai.com:

- The *EdgeSuite ESI/XSLT Development Tool* contains the procedures for running a debugger on both ESI and XSLT pages.
- The *ESI Test Server Users Guide*. You can run and test your ESI code on a test server before taking it live. The test server is in addition to ESID, the ESI Development Tool, which is covered in the last chapter of this document.
- *EdgeSuite Edge Transformations Service Overview* describes EdgeSuite's implementation of XSLT, which can be used in combination with ESI.
- The *EdgeSuite Configuration Guide* details the configuration and control options and parameters used for sites and objects in EdgeSuite, including ESI.
- For information specifically on cookies and Session IDs, see *EdgeSuite Session ID Support*.
- *EdgeSuite Handling of Edge-Control & Other HTTP Headers,* a discussion of the use of HTTP request and response headers in the EdgeSuite environment.
- *Time-to-Live in Cache: Methods and Considerations*. This discusses the various methods for determining the caching properties of objects on Akamai EdgeSuite servers.

CHAPTER 2. ESI Language Elements

A Quick Reference to the ESI Language

Table 1: ESI Language Elements

TYPE OF TASK	TO…	SEE…	PAGE
Object Inclusion	Create an include statement	include	15
Object inclusion with evaluation	Include objects with ESI code to be parsed in parent	eval	25
Conditional Inclusion	Add conditional processing	choose | when | otherwise	31
Iteration	Iteration through lists or dictionaries	foreach | break	34
Alternatives (to ESI) processing	Set alternative HTML to be used if ESI is not processed	remove	39
	Hide ESI statements if ESI is not processed	<!--esi -->	40
Place ESI code outside of ESI blocks	Put a variable or function outside an ESI block	vars	41
Exception Handling	Set exception handling statements	try | attempt | except	42
Comments	Add comments to code	comment	43
Variables	Use CGI variables	HTTP and Other Client Headers	45
	Use Cookies	Cookie Support	46
	Use POST responses	POST Support	46
	Use Akamai-Specific Variables	Akamai-Specific Variables	47
	Create variables using expressions	assign	51

Table 1: ESI Language Elements

TYPE OF TASK	TO…	SEE…	PAGE
Escaping characters	Using the backslash (\)	Escaping the $ and Other Reserved Characters	59
Boolean Operators	Evaluate conditions to true or false	Boolean Expressions	60
Expressions	Construct data structures of various types	Expressions	63
Logical operators	Evaluate Boolean and string expressions	Logical and String Operators	64
Functions	Generate response headers, cookies, random numbers, etc.	ESI Functions	67
Internationalization	Use dual- or multibyte character sets	Internationalization	87
Debug ESI pages	To send ESI pages through Debugger, use the < esi:debug/> tag, etc. See the document, *The ESI & XSLT Development Tool*.		

ESI Elements

About ESI Syntax

ESI elements and attributes are XML-based but can be embedded in other documents such as HTML or XML documents. EdgeSuite ignores everything except elements that begin with **`<esi:`** or **`<!--esi`** and terminate according to the syntax of the element. When EdgeSuite processes the page, the ESI elements themselves are stripped from the output.

ESI language syntax adheres to general XML syntax rules. Attributes can be arranged in any order within an ESI statement. ESI statements are case sensitive; ESI elements use lower case. ESI-supported CGI environment variables require upper case. The white space between the ESI sentence elements can be space characters, tab characters or new line characters.

Legend to Notation

Legend to Syntax Notation

`bold` = **required phrase**

`plain` = optional phrase or variable name

`italic` = user-supplied data

CHAPTER 3. Including Objects

The central ESI function is object inclusion, and the basic inclusion statement is **esi:include**. A second inclusion statement, **esi:eval**, allows you to include ESI evaluations that cannot be passed with **esi:include**.

The include Statement

include

The **include** statement provides for several optional attributes for alternative objects, error handling, caching, and dynamic processing. The include statement has a short form and a long form.

Short Form

The short form **include** statement is formulated as follows:

```
<esi:include src="object" attr1="val1" attr2="val2" etc./>
```

Where the optional attributes are listed in Table 2 on page 16. Two examples:

```
<esi:include src="http://www.akamai.com/frag1.html"
      alt="http://www.akamai.com/frag2.html" onerror="continue"
      maxwait="2000" ttl="4h"/>
<esi:include src="http://search.akamai.com search?query=
      $(QUERY_STRING{'query'})"/>
```

Long Form

The long form[1] is structured to pass XSL *params* when using Edge Transformations to perform an XSLT transformation on an object to use as an included fragment an ESI template.

```
<esi:include src="a.xml" attr1="val1" attr2="val2" etc.>
      <esi:param name="foo1" value="variable_1"/>
      <esi:param name="foo2" value="$var"/>
</esi:include>
```

In this form, the *attr1, attr2, etc.*, are any of the attributes used in the short form described in the preceding paragraphs. The long form is further described under "Long Form and XSL Params" on page 21.

The **include** statement is the essential statement in the ESI language. If **include** is successful, the contents of the **src** or **alt** URL replace the construct. The included object is included exactly at the point of the **include** statement; for example, if the **include** statement is in a table cell, the object is displayed in the table cell.

1. The long form of the **include** statement and the **esi:param name** tag are extensions to he ESI 1.0 specification. See page 8.

The fragments should be well formed: correctly assembled, syntactically correct, and contain the proper headers and Akamai metadata. The included object should be of the proper content type (*e.g.,* text/*) and should have no tags that conflict with or duplicate tags in the template document (such as additional <HEAD> tags, etc.).

Attributes

Of the attributes, only **`src`**, which specifies the main object to include, is mandatory. The attributes of the **`include`** statement are as follows:

Table 2: `include` Statement Attributes

ATTRIBUTE	DESCRIPTION	SEE PAGE...
src	The only *mandatory* attribute, the src is the primary object to fetch from the origin server or from cache when appropriate.	17
alt	An object to fetch if the src object is not found[a].	17
dca[b]	The type of processing for object. The default is "none."	17
onerror	The only argument, "continue," specifies ignoring failed fetches and continue serving the page without the results of the tag.	18
maxwait	A time-out period, in milliseconds, for EdgeSuite to wait for the src, alt, or stylesheet to complete the fetch successfully. This should be set to 1000 ms or higher in most cases.	18
ttl[b]	A time interval for the fetched object to reside in cache before EdgeSuite revalidates that the object has not changed.	18
no-store[b]	Turns on or off the instruction to EdgeSuite not to cache the object.	19
appendheader[b]	Adds a header to the HTTP request.	19
removeheader[b]	Removes all instances of a header from the HTTP request.	19
setheader[b]	Set the named header a specified value.	19
method[b]	Use GET (default) or POST method when fetching the src or alt object.	20
entity[b]	Used with a POST request method to set data to send with the POST.	20
stylesheet[b]	An XSL stylesheet to use with an XML object or alternate object in an XSLT transformation.	20
esi:param name[b]	Defines an XSL param to pass to XSLT processor. Used only with the long form, as discussed on page 21.	21

a. With regard to the **`alt`** option and the **`onerror`** attribute, "not found" or "failed fetch" means that EdgeSuite has received a non-200 series HTTP code response. For discussion of the conditions that trigger the use of a default object, see the discussion on page 95.

b. These attributes are extensions to the ESI 1.0 specifications. See "The ESI Specification" on page 8.

src and alt

The object specified by the **src** or **alt** can be a URL, as shown in Example 1; or, the object can include variables or a query string. The object can also be the result of EdgeSuite for Java processing or of an XSLT transformation.

The specified URL can be any domain, so long as the EdgeSuite configuration is set up to recognize the domain. The specified path can be relative to the host specified with the template— **/subdir/file.html** is as valid as **http://xyz.com/subdir/file.html** as a specification for a fragment when **xyz.com** is the host on which the template originated.

A query string—a question mark (?) followed by "key=value" pairs separated by ampersands (&)—can be added to the **src** or **alt** object. The string should be escaped—no "unsafe" characters such as spaces or brackets—because the string is sent exactly as it appears in the tag. The query string can contain additional question marks. You can use the **url_encode()** function, described on page 76. Here's an example of using the **url_encode()** function on an **alt** object that includes a query string that contains a variable **($(url))** that may contain unsafe characters:

```
<esi:include src="notfound.html"alt="hi.html?url=$url_encode($(url))"/>
```

You can also use the **esi:try** statement, described on page 42, to set an alternate object to fetch if the primary **include** fails. Use the **esi:try** statement if you want specify different attributes for the alternative and primary objects, since each object is specified with a separate **esi:include** statement. For example, you could set the **src** with a POST method and the **alt** with GET.

dca

DCA refers to Dynamic Content Assembly, and as an attribute it refers to the type of processing to be applied to the **src** or **alt** object. By default, an included object will be included as flat text. You can specify in your EdgeSuite configuration file to process a fragment in the same method as the template, or you can specify to process a fragment with a specific method such as ESI, XSLT, or None. The optional **dca="none|esi|xslt|java|akamaizer"** can be used in the **include** statement to specify a different type of processing: **xslt** for the EdgeSuite Transformations service, **java** to include an object processed through EdgeComputing for Java, **akamaizer** for the EdgeAkamaizer, or **none** for no **dca** processing.

You can daisy-chain processors by using the following syntax. Note the single quotes within the double-quotes around the "akamaizer->esi". This is necessary so that the right arrow (->) isn't seen by ESI to be the closing bracket.:

```
<esi:include src="obj.html" dca="'akamaizer->esi'"/>
```

This statement means first parse and process the object with the EdgeAkamaizer, then process the output as ESI. You can daisy-chain the Akamaizer, XSLT, and ESI processors; you can daisy chain up to five processors total, in any order.

onerror="continue"

If EdgeSuite can fetch neither the **src** object nor the **alt** object, it returns a 404 HTTP error with a simple error message—unless the **onerror** attribute is present. The **onerror** attribute can be used with an **src** only or with both an **src** and **alt** attempt. If **onerror="continue"** is specified and the **src** and **alt** fail to fetch the object, ESI deletes the **include** tag and serves the page without any object replacing the **include** statement.

When **onerror="continue"** is set and the fetch fails, EdgeSuite does not serve a default object. Without the **onerror** attribute, EdgeSuite attempts to fetch a default object if one is specified in the configuration file.

For more information on error handling, see "Exception and Error Handling" on page 95. For information on using **onerror** inside ESI's explicit exception handling method, the **try** block, see the discussion beginning on page 43.

maxwait

This attribute sets a time-out period, in milliseconds, for EdgeSuite to wait for completion of the **src**, **alt**, or **stylesheet** fetch. If the time-out period expires, EdgeSuite moves to whatever comes next: delivering an error, processing based on a **onerror="continue"** attribute, attempting to fetch the **alt** object after an **src** failure, etc. If there is no maxwait specified, the default time-out period is 30 seconds. In some cases, it may not be desirable to use **maxwait** in an **esi:try** block (see page 43).

This should be set to 1000 ms or more in most cases, and particularly when the fragment being fetched requires processing. For example, fetching a fragment that itself will receive XSLT transformation takes time, and more than .5 seconds for the fetch, compile stylesheet, and transform to take place is not unreasonable.

Using maxwait = 0 to Send an Asynchronous include

A value of 0 on the **maxwait** attribute has a special meaning: "send the request but do not wait to receive the object." This allows you to set up a "delivery receipt" condition, also known as an "asynch include:" you can notify the origin server that the object has been requested.

An asynch include should also include the **onerror** attribute:

```
<esi:include src="somefile.html maxwait="0" onerror="continue"/>
```

In most cases, you'll probably not want to put a positive TTL on **somefile.html**. If EdgeSuite simply has to retrieve the object from cache, the origin may not be notified of the request.

ttl

This specifies a Time-To-Live (TTL) for the object to be stored in EdgeSuite's cache—the maximum amount of time the content will be served before EdgeSuite issues an **If Modified Since** (IMS) request to the origin server to check whether the object content has changed. EdgeSuite issues an IMS only if the object is requested.

The **ttl** specifies a time-to-live in cache for the source files, not for ESI or XSLT results. It is the *source* object requested from the origin server that is cached according to the **ttl** instructions. By default, the ESI resulting objects are not cached, but you can set up result caching through EdgeSuite configuration.

The value is an integer, 0 or greater, followed by a unit specifier. A **ttl=0s** means that the object is cached but EdgeSuite will revalidate it every time it is requested.

The unit specifier can be *one* of the following: **s** (seconds), **m** (minutes), **h** (hours), or **d** (days). The specifiers cannot be combined—**120m** is OK, but **1d4h20m** is not.

This settings from this attribute override any other ttl settings from other sources such as the EdgeSuite configuration file, the Edge-control header, or HTTP headers. The exception is that the **ttl** does not override a **no-store**.

If the attribute is not set, the default time-out will be a default time-out for your configuration, if one is set, or if none is set, the time-out period for the Akamai server. Check with your Akamai representative for details, if necessary.

no-store

A **no-store="on"** tells EdgeSuite not to cache the object, and it overrides other caching instructions such as **ttl**. A **no-store="off"** turns off the **no-store** instruction.

appendheader, removeheader, setheader

These three headers add, remove, or set headers that accompany the HTTP request to fetch the **src** or **alt** object:

- **appendheader** adds a new header. The format and an example:

```
appendheader="field-name: field-value"
appendheader="Accept-charset: iso-8859-5"
```

- **removeheader** removes all instances of a specified header.

```
removeheader="field-name"
removeheader="If-unmodified-since"
```

- **setheader** sets the named header to a specified value. Conceptually, this is the equivalent of a **removeheader** plus an **appendheader.** Unlike **appendheader, setheader** can modify existing headers.

```
setheader="field-name: field-value"
setheader="Content-type: Text/*"
```

xxxheader Attributes Accept Expressions for Names or Values

These attribute all accept ESI variables or other legitimate ESI expressions (see page 63) as the field-name or field-value. For example, the following construction might be used where **a_name** and **a_value** are legitimate variables or other ESI expressions:

```
<esi:include src="obj.htm" appendheader="$(a_name) + ':' + $(a_value)"/>
```

Usage Principles

Other usage principles are as follows:

- A single **include** statement can contain multiple instances of any of these header attributes. For example, this is OK:

```
setheader="a_header: a_value", setheader="b_header: b_value"
```

- **appendheader** can be used multiple times with the same header field-name. For example, this is OK:

```
appendheader="a_header: value1", appendheader="a_header: value2"
```

- This latter construction is *not* OK for **setheader** or **removeheader**. For these two attributes, the same field-name should be used only once in an **include** statement.

  The following example is not a valid construction; while it won't generate an error, it may yield unpredictable results:

  ```
  setheader="a_header: value1", setheader="a_header: value2"
  ```

- The ***processing order*** for these attributes in an **include** statement is as follows: first, all the **removeheader**s are performed, then all the **setheader**s, then all **appendheader**s. If, for example, you remove a header with **removeheader** and set it but with a different value with **appendheader**, all in the same include statement, the net result is that the value set with the **appendheader** prevails.

method

This specifies the use of a GET or POST method for the HTTP request sent to fetch the **src** or **alt** object. The default is GET. The format is:

```
method= "GET | POST"
```

For example, **method="POST"**.

This attribute can accept an expression, for example, **method="$(var_method)"**

entity

This sets the entity, the message body, to send with a POST request, and is used only when the **method** is POST. It is ignored when the method is GET. The format is:

```
entity="message body"
```

This attribute accepts ESI variables or other expressions as the value. For example:

```
<esi:include src="foo" method="POST" entity="$(QUERY_STRING)"/>
```

Note that the following two constructions will send the same data in the request:

```
<esi:include src="foo?x=7"/>
<esi:include src="foo" method="post" entity="x=7"/>
```

The first statement uses the default GET method and a query string. The second statement uses a POST with a message body. Both are valid ways to perform requests.

stylesheet

The optional **stylesheet** attribute is used only when specifying an XSL stylesheet to use with an XML object or alternate object in an XSLT transformation. The form is **stylesheet="*foo.xsl*",** where *foo.xsl* is the stylesheet. The stylesheet must come from the same host name as the **src** or **alt** object, or your EdgeSuite configuration must be set up to allow for a stylesheet from any host.

If the **src** or **alt** object XML file is not processed by Edge Transformations, the **stylesheet** attribute is ignored. A stylesheet specified here takes precedence over stylesheets specified in the XML document, but the stylesheet does not apply to any nested XSL stylesheets contained within the stylesheet itself.

The **maxwait** attribute applies to the **stylesheet**, but the **onerror**, **ttl**, and **no-store** and other attributes affect only the **src** or **alt** objects. With regard to the **dca** attribute, **stylesheet** applies only when **dca="xslt"**.

Integration with XSLT

Several **include** features and other ESI language features are geared to facilitate integration with EdgeSuite Edge Transformations, allowing you not only to include XSLT transformed objects as fragments in ESI constructions, but also to control caching, pass parameters from ESI to XSLT, and so forth. In summary, these integration functions are as follows:

- The **dca="xslt"** attribute allows you to specify XSLT parsing and transformation for a fragment.
- The **stylesheet** attribute allows you to specify an XSL stylesheet to use to transform the XML object.
- The **esi:param name** attribute allows you to pass XSL *params* when performing XSLT processing. This attribute is used with the long form of the **include** statement, discussed in the next subsection.
- The other **include** attributes, such as **ttl**, **no-store**, **maxwait**, and **onerror**, can be used to control properties of the XML object, but not the XSL stylesheet.
- Care should be taken when setting the maxwait attribute; values greater than 1000 ms should be used, if this attribute is to be used, since fetching XML and stylesheet, compiling the stylesheet, and transforming the file can all take time.
- You can pass HTTP headers/CGI variables and Akamai-specific variables such as EdgeScape data to XML files included as ESI fragments. Variables are discussed in "ESI Variables Support" on page 45.
- The EdgeSuite ESI/XSLT Development Tool (debugger) produces a debug report that can include both ESI and XSLT processed pages, including XSLT fragments in ESI documents. This is covered in a separate document, the *EdgeSuite ESI/XSLT Development Tool*.

For further information on the XSLT service, see the *EdgeSuite Edge Transformations Service Overview*.

Long Form and XSL Params

Use the long form of the include statement to pass XSL *params* when including an XSLT transformed fragment. In combination with the **assign** statement (to create user-defined variables, described on page 51), you can first create and then pass parameters.

param name

For example:

```
<esi:assign name="var1" value="'variable_2'"/>
<esi:include src="a.xml" dca="xslt" stylesheet="s.xsl">
      <esi:param name="foo1" value="variable_1"/>
      <esi:param name="foo2" value="$var1/>
</esi:include>
```

The value defined in the **param name** statement can be any legal ESI expression, as described in Table 5 on page 63. The data type of the value passed is always a string. This example shows two params defined. The first is defined as a string variable, *variable_1*. The second is defined as the evaluation of the user-defined variable, *var1*, so that the string value, *variable_2*, will be passed to *a.xml*.

In the long form, the only allowable statements between the opening and closing tags are the **esi:param name** statement. When you use include statement attributes, place them into the opening tag. For example:

```
<esi:include src="a.xml" alt="b.xml" stylesheet="s.xsl" maxwait="2000"
      dca="xslt">
      <esi:param name="foo1" value="variable_1"/>
</esi:include>
```

Attributes placed inside the block may create errors; other text inside the block will be ignored. For example:

```
<esi:include src="a.xml" dca="xslt">
      this text will be ignored
      <esi:param name="foo1" value="variable_1"/>
      <esi:any tag other than param name will cause an error/>
</esi:include>
```

In the case of *param* name conflicts, the last definition is used. For example, in this situation, the second value, *bar2*, would be passed to the XSLT processor:

```
<esi:include src="a.xml" dca="xslt">
      <esi:param name="foo" value="bar1"/>
      <esi:param name="foo" value="bar2"/>
</esi:include>
```

Don't Pass Apostrophes to XSLT

When passing parameters or queries to XSLT, don't pass apostrophes, since they are used as delimiters in XSLT parameters. You can replace apostrophes with "_" or "'", and change your XSL stylesheet accordingly.

Controlling Downstream Caching

For more information on caching, see the document, *Time-to-Live in Cache: Methods and Considerations.*

Setting a positive value, as opposed to a no-store or no-cache, for downstream caches should be done only when it's appropriate to the content. For example, if you're using ESI to build user-customized pages, you probably won't want to set a positive downstream ttl unless you are certain the pages won't be cached in proxy server and then served to the wrong user. To use an opposite example, if you're using ESI to build "latest update" pages for game scores and statistics, you might want the downstream caches to be able to store the end-of-game-night scores and statistics until the beginning of games the next day.

Since every template and fragment has its own caching properties in EdgeSuite, the following logic is used to control the downstream caching for ESI produced pages.

- If a **no-store** is present, it will be propagated up to the root response header returned to the client.
- Otherwise, if a **ttl** is present, the value will apply to the object to which it is associated, and in addition, the root response header will be set to no more than the

least **ttl** value of all objects associated with the template. In other words, the resulting ESI page will be cached downstream for a period defined by the shortest **ttl** of all the objects that compose the page.

Using Response Headers

You can use an Edge-control response header to set a downstream TTL for ESI results.

```
Edge-control: downstream-ttl=30m
```

The value is an integer may be followed by a unit specifier: Current unit specifiers are: 's' (seconds), 'm' (minutes), 'h' (hours), 'd' (days). The default is ‘s’.

Using the Configuration

You can use your EdgeSuite Configuration file to set a downstream TTL for ESI results. When set on a fragment, this value is used as one of the values used by ESI to calculate the TTL value as defined in the preceding paragraph.

When you set a downstream TTL value in this manner on the template page, this value overrides TTL settings assigned with include attributes or in your configuration file. However, the configuration setting does not override a no-store or bypass-cache, nor does it take precedence over a value set with the **`downstream-ttl`** Edge-control header.

When you allow ESI to calculate the downstream TTL, the value it sends will account for time already spent in cache. But when you use the Downstream TTL attribute, neither Cache-control: max-age setting nor the Expires data can be updated to account for time already spent in cache.

Also, you can use the **`$add_cachebusting_header()`** to prevent downstream caching, as discussed on page 76.

Secure and Not Secure

If the template is a secure URL under the SSL protocol, then all fragments must be secure. If the template is not secure, the fragments can be either secure or not. That is, you can upgrade the security level by including secure fragments in an unsecured template, but you cannot downgrade by included unsecured fragments in a secure template.

Limitations

EdgeSuite ESI supports up to fifteen levels of nested **`include`** statements: that is, fragments can contain ESI markup that includes other fragments. The first **`include`** is the first of the levels.

Also, ESI provides for up to sixty-five attempted **`include`**s per transaction. The transaction begins with the first **`include`** statement on the template and encompasses all subsequent **`include`**s associated with that template, including the attempts generated in nested pages.

If the **`src`** object is fetched and there is no need to attempt the **`alt`**, that is one attempt; if the **`src`** include fails and the **`alt`** is fetched, that is two attempts. When it processes a page, EdgeSuite processes all **`src`** objects before it processes any **`alt`** objects.

The total size of all included objects, including nested objects, for a page, cannot exceed 1 megabyte.

Evaluating Included Objects

eval

On the surface, the **esi:eval** statement is similar to the **esi:include** statement. They both fetch fragments, employ an almost identical syntax, and accept almost the same attribute set (page Table 2 on page 16).

The main difference is that while **esi:include** fetches the fragment and inserts it verbatim into the template, **esi:eval** interprets the fragment as ESI code running within the execution context of the template—as if the fetched object's code is in the template itself. You can think of **eval** as fetching the fragment and pasting it into the template *before* the ESI code is interpreted.

For example, you can store a cookie on the end-user's machine that contains a customer number. When you receive a request from the end-user's browser, you could retrieve data from your database and use the data in an ESI page to display it to the end-user.

This is significant because with **include**, variables pass from templates to fragments (from parent to child), not the other direction. It is impossible for a child to affect a parent's *namespace*—the attributes, variables, and other elements associated with the template as identified by its URL. However, the **esi:eval** statement runs in the parent's namespace, so that the namespace can be changed by **eval** processing.

To illustrate, take the contents of a file used as an ESI fragment, **cust_1234.html**:

```
<esi:assign[a] name="first" value="Gene"/>
<esi:assign name="last" value="Kranz"/>
<esi:assign name="age" value="67"/>
<esi:assign name="state" value="CA"/>
```

a. **esi:assign**, which creates a user-defined variable, is described on page 51.

The template, **welcome.html**, contains the following ESI code. In this example, the variable, **C_URL**, evaluates to **cust_1234.html**.

```
<esi:assign C_URL="'cust_' + $(HTTP_COOKIE{CustomerCode}) + '.html'"/>
<esi:eval src="$(C_URL)" dca=”none”/>
<esi:vars>
      Welcome $(first), how is the weather in $(state)?
</esi:vars>
```

When requested by the person with the **CustomerCode** cookie set to 1234, the client receives the following output:

```
Welcome Gene, how is the weather in CA?
```

The **eval** statement allows the *use* of the variables assigned in the fragment, not simply the inclusion of the page as a object.

Furthermore, since variables are still automatically passed from parents to children, you can pass variables from the template to the fragment and then modify the variables when they are included by **eval** into the template from the fragment.

Usage and Restrictions

As noted above, the **esi:eval** statement accepts most of the same attributes as **esi:include**. The following differences or unique considerations should be noted:

- The **eval** does *not* use the **alt** attribute. You can use the **try** block to specify objects to fetch if the **src** object cannot be fetched. Note that the **try** block provides you more control than the the **alt** syntax, since you can specify different attributes for the secondary objects.
- A maximum of 10 **eval** statements can be used on a template and its fragments.

Special Processing

The **eval** statement provides exceptions to ESI processing rules in two ways:

- As discussed on the previous page, since the **eval** fragment is processed in the parent's namespace, it can modify and create variables in that namespace.
- Generally, variables set inside an **esi:try** block (page 42), a conditional block used to set exception processing, cannot be used outside the **try** block. However, when the **eval** statement is used in the successful completion of the **try** block, evaluations can be used in the rest of the body of the parent.

  For example, take the fragment, **frag.html**, containing this ESI code:

  ```
  <esi:assign name="NAME" value="Joe"/>
  ```

  Now note the template, **template.html**, containing the **try** block:

  ```
  <esi:try>
      <esi:attempt>
          <esi:eval src="frag.html" dca="none"/>
      </esi:attempt>
  </esi:try>

  <esi:vars name="$(NAME)"/>
  ```

  Once run, **$(NAME)** outputs **Joe**. The variable created in the fragment is used in the template, and it can be used outside the **try** block in which it was fetched. If you had used **esi:include** instead of **esi:eval**, the output would have yielded an error—the variable **NAME** would not have been found.

  Note, however, that the evaluation cannot be used inside the **try** block. In the above example, **$(NAME)** would not be a valid variable inside the try block.

Performance Considerations

Every time the ESI processor finds an **eval** statement in your code, it must stop and wait for the fragment to be retrieved so that the code can be included in the namespace. The **eval**s in your code must be processed serially—for two or more evals in a page, the order can be important. This differs from **include**s, which are all fetched at the same time and then inserted whenever they come back.

Furthermore, every **esi:eval** has the cost of an **esi:include** *plus* the overhead of processing the ESI code.

Overall, the performance cost on the **eval** can be mitigated by caching, but still it can be significant, depending on the situation. Certainly, pages with a large number of **eval** statements will see a significant slowdown in processing speed. On the other hand, there are times when a problem can only be solved through the use of **eval**.

Errors

With the **eval** statement, errors can occur in three places:

- Fetching the fragment. Errors are indicated by the response code from the fetch, exactly like an **include**.
- Parsing the fragment. This results in an HTTP 500 series error and is listed as a syntax error in the Debugger report.
- Executing the fragment. This results in an HTTP 500 series error and is listed as an error in the Debugger report.

For more information on error processing in ESI, see page 95.

The Importance of the dca Attribute in esi:eval

When a fragment's ESI code is processed in the parent's context, the results can vary dramatically depending on the type of dynamic content assembly (DCA) processor applied to the operation. The **dca** attribute, described on page 17, controls how the fragment is processed *before* it reaches the parent's context. For example, **esi** indicates that it should be processed by an ESI processor before inclusion in the parent, and **none** indicates that no processing should occur before inclusion.

By default, the processor is "none"—that is, the fragment will not be parsed for ESI, transformed by XSLT, etc. However, your EdgeSuite configuration can be set to automatically use the same processor for the fragment that was used for the template.

Comparing Processor Types When Using eval

To illustrate the significance of the processing type, we can compare examples:

Example 1: dca="none"

Assume you have a fragment, **frag1.html**:

```
<esi:assign name="fvar" value="9"/>
<esi:assign name="pvar2 value="0"/>
```

The template, **parent.html**, follows. Note that one variable, **pvar2**, is defined in both the parent and the child. The parent's **eval** statement uses **dca="none"**.

```
<esi:assign name="pvar1" value="7"/>
<esi:assign name="pvar2" value="8"/>
<esi:eval src="frag1.html dca="none"/>
<esi:vars>
pvar1 = $(pvar1)
pvar2 = $(pvar2)
fvar  = $(fvar)
</esi:vars>
```

Here is a representation of the evaluation processing. The fragment's code, which is underlined and in **blue**, replaces the **eval** statement at the **eval**'s insertion point:

```
<esi:assign name="pvar1" value="7"/>
<esi:assign name="pvar2" value="8"/>
<esi:assign name="fvar" value="9"/>
<esi:assign name="pvar2 value="0"/>
<esi:vars>
pvar1 = $(pvar1)
pvar2 = $(pvar2)
fvar  = $(fvar)
</esi:vars>
```

And when **parent.html** is executed, the following output is produced.

```
pvar1 = 7
pvar2 = 0
fvar  = 9
```

Example 2: dca="esi"

In this example, the fragment is the same and the only difference in the parent is that in the **eval** statement, **dca** is now set to **"esi".** When **dca="esi",** the fragment is first processed in a different context, and the result is then executed by the parent. The ESI output of **frag1.html** is *nothing* (except for a few newline characters), so that "pasting" the **eval** processing into the parent, the parent looks like this:

```
<esi:assign name="pvar1" value="7"/>
<esi:assign name="pvar2" value="8"/>

<esi:vars>
pvar1 = $(pvar1)
pvar2 = $(pvar2)
fvar  = $(fvar)
</esi:vars>
```

Now when the parent is executed, the resulting output is very different from Example 1. There is no value for **fvar**, the variable set only in the fragment, and **pvar2** holds the value set in the parent.

```
pvar1 = 7
pvar2 = 8
fvar  =
```

Example 3: Adding <esi:text> to Examples 1 and 2

We change **frag1.html** in one minor way—we put the **assign** statements inside an **esi:text** block. The **esi:text** block, described on page 40, provides the ability to insert flat text without ESI processing.

The **frag1.html** file now looks like this:

```
<esi:text>
<esi:assign name="fvar" value="9"/>
<esi:assign name="pvar2 value="0"/>
</esi:text>
```

Now, if you execute the parent with **dca= "none"**, the fragment is "pasted" into the parent exactly as it appears above. Once again, the fragment code is highlighted.

```
<esi:assign name="pvar1" value="7"/>
<esi:assign name="pvar2" value="8"/>

<esi:text>
```

```
<esi:assign name="fvar" value="9"/>
<esi:assign name="pvar2 value="0"/>
</esi:text>

<esi:vars>
pvar1 = $(pvar1)
pvar2 = $(pvar2)
fvar  = $(fvar)
</esi:vars>
```

And the output is:

```
<esi:assign name="pvar1" value="7"/>
<esi:assign name="pvar2" value="8"/>

pvar1 = 7
pvar2 = 8
fvar  =
```

That is, the fragment's lines are treated as flat text, not ESI statements, so the variables are not assigned, and the **assign** lines show up in raw form in the output.

However, if you set **dca="esi"**, the **esi:text** block is processed, and while the strings inside the block are left intact, the **esi:text** opening and closing statements are removed, so that the parent including the **eval** looks like this:

```
<esi:assign name="pvar1" value="7"/>
<esi:assign name="pvar2" value="8"/>

<esi:assign name="fvar" value="9"/>
<esi:assign name="pvar2 value="0"/>

<esi:vars>
pvar1 = $(pvar1)
pvar2 = $(pvar2)
fvar  = $(fvar)
</esi:vars>
```

And the output is:

```
pvar1 = 7
pvar2 = 0
fvar  = 9
```

The difference is that the output under **dca= "esi"** is due to insertion of the ESI code after it was processed. Similarly, all other valid **dca** types can be used. For example, you might want to create ESI code from XSLT.

Using eval in Dynamic Code Generation

The **eval** statement provides a limited function-like ability. Consider a situation in which you want to make a copy of a dictionary. Simply assigning it to another variable creates a reference, not a copy, and a reference may not be what you need in a particular situation (see the discussion on page 55).

Without **eval**, you could create a copy by performing an iteration through the dictionary, illustrated by the following code for a dictionary, **dict**, and a copy, **copy**:

```
<esi:foreach collection="$(dict)">
```

```
    <esi:assign name="copy{$(item{0})}" value="$(item{1})"/>
</esi:foreach>
```

Rather than putting this on all of your ESI pages, you can use an **eval** statement with **dca="none"** to include it as a fragment. You could do the following, placing the code above into a fragment named **eval-copy-dict.html**:

```
<esi:assign name="dict" value="my_dict"/>
<esi:eval src="eval-copy-dict.html" dca="none"/>
<esi:assign name="my_copy" value="copy"/>
```

However, this solution requires you to know the names of the variables the fragment uses and that you limit yourself to a dictionary called **dict** and a copy called **copy**.

A more flexible solution would involve setting up a fragment that looks like this:

```
<esi:assign name="from" value="$(QUERY_STRING{from})"/>
<esi:assign name="to"   value="$(QUERY_STRING{to})"/>

<esi:vars>
\<esi:foreach collection="\$($(from))">
\<esi:assign name="$(to){\$(item{0})}" value="\$(item{1})"/>
\</esi:foreach>
</esi:vars>
```

Now you can call the fragment like this:

```
<esi:eval src="eval-copy-dict.html?from=my_dict&to=my_copy/>
```

CHAPTER 4. Conditional Inclusion and Iteration

This chapter covers the ESI iteration capability and the method ESI provides for qualifying the inclusion of objects.

Conditional Processing

choose | when | otherwise

The **choose** block provides a **when | otherwise** statement set, comparable to the *if-then / else* mechanism in some languages, allowing you to test for a logical true (value 1) or false (value 0). Boolean expressions are fully defined under "Boolean Expressions" on page 60. You can perform evaluations using a variety of environment variables, Akamai-specific and user-defined variables, and functions, described in subsequent chapters.

The **choose** block is formed as follows:

```
<esi:choose>
    <esi:when test="BOOL-expr">
        Do something
    </esi:when>
    <esi:when test="BOOL-expr">
        Do something different
    </esi:when>
    <esi:otherwise>
        Do something else…
    </esi:otherwise>
</esi:choose>
```

Example:

```
<esi:choose>
    <esi:when test="$(HTTP_COOKIE{'group'})=='Advanced'">
        <esi:include src="http://www.akamai.com/advanced.html"/>
    </esi:when>
    <esi:when test="$(HTTP_COOKIE{'group'})=='Basic User'">
        <esi:include src="http://www.akamai.com/basic.html"/>
    </esi:when>
    <esi:otherwise>
        <esi:include src="http://www.akamai.com/new_user.html"/>
    </esi:otherwise>
</esi:choose>
```

Usage

- Each **choose** block and contained sub-blocks must be closed via a separate close tag. Thus, the syntax **<esi:choose…/>** is in*valid.*
- Each **choose** block must have at least one **when** element.
- The **otherwise** element is optional.

- The **when** tag is synonymous with **if** and **else-if** in other languages. It evaluates the Boolean expression using the test attribute.
- Only the first **when** clause evaluated as true is executed.
- The **otherwise** tag is synonymous with **else** in other languages. The **otherwise** clause is evaluated only if all **when** clauses evaluate to false.
- To test for an empty or non-existent variable, use the following construction: **<esi:when test="$(foo)">**, where "foo" is the variable being tested. If the variable is empty or non-existent, the test returns "false." Note that there are also functions, **$is_empty()** and **$exists()**, discussed on page 74, to test precisely for empty or non-existent variables, that can be used in the **choose** block.
- Regular HTML and ESI statements can be included inside **when** or **otherwise** statements. However, do not place these statements *outside* a **when** or **otherwise** sub-block. This is further explained in the next subsection.

Compound Expressions

You can use any legal ESI expression or compound expression in the **when** evaluation, using variables, functions, and Booleans described in other chapters of this guide.

For example, you could construct a choose statement as follows:

```
<esi:choose>
  <esi:when test="!$exists($(HTTP_COOKIE{'UserInfo'})) |
            !($(HTTP_COOKIE{'UserInfo'}) matches '''UserId=[0-9]''')">
      [include some file]
  </esi:when>
  <esi:otherwise>
      [include some other file]
  </esi:otherwise>
</esi:choose>
```

The **<esi:when>** statement shows the use of a compound expression using functions and variables combining several different operators. It uses the Booleans **!** (not) and Or (|), and a regular expression (regex) match (**matches**). It uses the variable and variable substructure **$(HTTP_COOKIE{UserInfo})**. And it uses the function **$exists** to check to see if the cookie exists.

The statement reads as follows: IF the cookie does NOT EXIST, OR if the cookie EXISTS but does NOT contain a regex match on the specified UserID data, THEN include "some file." OTHERWISE, include "some other file."

Statements Inside a Block

In a **choose** construct, valid statements must be placed inside a **when** or **otherwise** sub-block. Statements outside the sub-blocks are not evaluated as conditions. In the following example, the lines marked with daggers (†) are invalid and will be discarded by the EdgeSuite processor.

```
<esi:choose>
      † Invalid HTML here
      <esi:when>
          <esi:include...>
          This line is valid and will be processed.
      </esi:when>
      † Invalid HTML here
      <esi:otherwise>
          <esi:include...>
          This line is valid and will be processed.
      </esi:otherwise>
      † Invalid HTML here
</esi:choose>
```

Valid statements can be more than one line, and do not need to be **include** statements shown in the previous example. The following example tests to see if an Akamai variable (the GEO region code) is empty, and if it is, set a cookie and redirect the request. (The functions are discussed in "ESI Functions" on page 67.)

```
<esi:choose>
  <esi:when test="$isempty($GEO{'region_code'})">
      $set_redirect('no_region.myplace.com')
      $add_header('Set-Cookie', 'MyCookie1=SomeValue;')
  </esi:when>
  <esi:otherwise>
      ...more valid code....
  </esi:otherwise>
</esi:choose>
```

Nesting Elements

Most ESI elements may contain child elements, including nested ESI elements and other markup. This allows block structuring of compound conditionals. The following construct shows a nested **choose** block, **bolded** here for clarity:

```
<esi:choose>
          <esi:when test="BOOL-expr">
              <esi:include src="URL-expr"/>
          </esi:when>
          <esi:when test="BOOL-expr">
              <esi:choose>
                  <esi:when test="BOOL-expr">
                      <esi:include src="URL-expr"/>
                  </esi:when>
                  <esi:otherwise>
                      <esi:include src="URL-expr"/>
                  </esi:otherwise>
              </esi:choose>
          </esi:when>
          <esi:otherwise>
              <esi:include src="URL-expr"/>
          </esi:otherwise>
</esi:choose>
```

Iteration

foreach | break

The **foreach**[1] statements provides the ability to iterate through a sequence. The block is formed as follows:

```
<esi:foreach item="<item>" collection="<sequence>">
  <ESI code goes here>
</esi:foreach>
```

If you do not specify a name for **<item>**, it will default to the name "item."

In each iteration, a single item from the collection is available under the name **$(<item>)**, where **<item>** can be a user-specified string or a dictionary. Strings, dictionaries, and lists are expressions defined on Table 5 on page 63.

When the item is a dictionary, each **$(<item>)** is a list. The term **$(item{0})** is the *name* of the dictionary entry, and **$(item{1})** is its *value*. However, there is no way to determine the order in which pairs will be processed.

Modifying **<sequence>**—for example, deleting the item processed—has no effect on the number of iterations, as a copy of the sequence is used to iterate over, to prevent runaway loops.

By default, the ability to use the **foreach** statement is enabled in ESI, but you can disable it using a setting in the EdgeSuite configuration file.

The foreach statements can be nested inside one another. There is a limit of 1,000 iterations for a statement, and the resulting ESI objects cannot be greater than 500,000 bytes in size.

Lists and Dictionaries

Example of a simple list iteration

```
<esi:foreach item="My_Item" collection="[1,2,3,4,5]">
  This is iteration number $(My_Item)<br>
</esi:foreach>
```

This would yield the text:

```
This is iteration number 1
This is iteration number 2
........iteration number 5
```

Example of a simple dictionary, a list of fruits:

```
A list of Fruits:
<esi:foreach
      collection="{1:'apples',2:'oranges',3:'bananas',4:'grapefruits'}"
$(item)  --  $(item{0}) = $(item{1})<br>
</esi:foreach>
```

In this example, note that since there is no **item** attribute specified, the value defaults to "item."

1. **foreach** is in addition to the ESI 1.0 specifications. See "The ESI Specification" on page 8.

The example would yield:

```
A List of Fruits:
(4, 'grapefruits')  --  4 = grapefruits<br>
(3, 'bananas')  --  3 = bananas<br>
(2, 'oranges')  --  2 = oranges<br>
(1, 'apples')  --  1 = apples<br>
```

Note that the display order is not determined by the order of items in the dictionary.

A query string is a form of dictionary. For example, using the query string **"?a=1&b=2&c=3&d=4&aa=5&aab=6"**:

```
The QUERY_STRING is:
<esi:foreach item="query_pair" collection="$(QUERY_STRING)">
  $(query_pair{0}): $(query_pair{1})<br>
</esi:foreach>
```

Once again, the processing order is not predetermined. This could yield:

```
The QUERY_STRING is:
aab: 6<br>
d: 4<br>
b: 2<br>
c: 3<br>
aa: 5<br>
a: 1<br>
```

Iterating Over a Range

You can use the range operator to iterate a specific number of loops or to create a list of a known number of items. The following are valid constructions:

```
<esi:foreach collection="[1..10]">
<esi:foreach collection="[5..1]">
<esi:foreach collection="[10 .. $(max)]">
```

See page 65 for more information on the range operator.

Breaking the Loop

The optional **break** statement can be used to exit early from the closest enclosing iterative loop. Its presence in anywhere other than inside a **foreach** block will create a syntax error.

For example:

```
<esi:foreach collection="[1,2,3,4,5]">
  <esi:choose>
    <esi:when test="$(item) == 3">
     <esi:break/>
   </esi:when>
 </esi:choose>
 <esi:vars>$(item) </esi:vars>
</esi:foreach>
```

This statement prints out: "1 2 ". The loop exits when the third element is reached.

Another example:

```
<esi:foreach item="bar" collection="[1,2,3,4,5]">
  <esi:foreach item="foo" collection="['a', 'b', 'c', 'd', 'e']>
    <esi:comment value="This break exits the 'foo' loop, not the 'bar'
      loop/>
    <esi:break/>
    $(foo)
  </esi:foreach>
  $(bar)
</esi:foreach>
```

Iteration Index Keys

You can also use the following index keys:

- **<item>_index**—a zero offset integer specifying the index of the current item. The first item has index value of 0, second is 1, and so forth.
- **<item>_number**—a 1 offset integer that behaves just like **_index**. This may process more quickly than **_index**, because it takes advantage of a faster math processing algorithm.
- **<item>_start**—is true only when in the first iteration
- **<item>_end**—is true only when in the last iteration
- **<item>_odd**—is true whenever the iteration number is odd. The first iteration is odd, second even, and so forth.
- **<item>_even**—is true whenever the number is even.
- **<item>_sequence_size**—always contains the total length of the sequence, which does not change with each iteration.

Example Using the Index Keys and a POST Response

A simple example of using the keys to iterate over POST response data, using embedding **esi:choose** blocks to qualify its processing and formatting, might be as follows: The user selects items to order. You want to respond confirm the item numbers.

The data is contained in a POST response, a dictionary in ESI (see page 46). In this case, the POST response contains the following dictionary: **123:'Rocking**

Chair',124:'Sofa',678:'Microwave Oven',910:'Big Screen Television', 408:'Power Drill',650:'Bench Saw'.

```
<esi:foreach item="My_Item" collection="$(POST)">
<esi:choose>
    <esi:when test="$(My_Item_start)">
        Here is the contents of your shopping cart
      ($(My_Item_sequence_size) Items):<br>
        <table>
        <tr><td>Entry #</td><td>Product #</td><td>Description</td></tr>
    </esi:when>
</esi:choose>
<esi:choose>
    <esi:when test="$(My_Item_odd)"><tr bgcolor="white"></esi:when>
    <esi:otherwise><tr bgcolor="silver"></esi:otherwise>
</esi:choose><td>$(My_Item_number)</td><td>$(My_Item{0})</
      td><td>$(My_Item{1})</td></tr>
<esi:choose>
    <esi:when test="$(My_Item_end)">
        </table>
        End of Shopping cart contents
    </esi:when>
</esi:choose>
</esi:foreach>
```

This is the result. Once again, note the item order in the dictionary doesn't predict the order of the displayed results:

Here is the contents of your shopping cart (6 Items):

Entry #	Product #	Description
1	910	Big Screen Television
2	678	Microwave Oven
3	124	Sofa
4	123	Rocking Chair
5	650	Bench Saw
6	408	Power Drill

End of Shopping cart contents

CHAPTER 5. Alternative Processing and Exception Handling

This chapter covers alternative processing, placing variables and functions outside of ESI blocks, inserting pure text, and handling exceptions. Also covered here is the comment statement.

Including Alternative HTML and Hiding the ESI Statements

<!--esi --> and remove

You can use the **remove** statement to include alternative HTML markup that browsers can display in the event ESI processing cannot be performed. In addition, you can use the **<!--esi** and **-->** tags to hide ESI statements in the event the content of the page is passed unprocessed to the browser.

remove

The **remove** statement provides for including valid HTML as output if the ESI markup is unprocessed, but removes the content if the markup is processed normally.

```
<esi:remove>
    Some HTML goes here…
</esi:remove>
```

Example:

```
<esi:include src="http://www.akamai.com/ad.html"/>
<esi:remove>
     <a href="http://www.akamai.com">www.akamai.com</a>
</esi:remove>
```

Description

Normally, when EdgeSuite processes this example block, it fetches the **ad.html** file and places it into the template page while discarding the **remove** block. If for some reason this block is not processed by EdgeSuite and all four lines are sent directly to the browser, all ESI elements are ignored by the browser and, most significantly, the **href** HTML statement is rendered by the browser.

This works simply because the browser throws away the HTML invalid **<esi:...>** and **</esi:...>** tags, leaving the HTML-valid **href** statement.

The remove statement cannot include nested ESI markup, and it cannot be included inside other ESI markup—for example, it cannot be used inside a **choose** block.

<!--esi -->

The **<!--esi -->** block allows for all ESI language to be hidden if the contents of the page are passed unprocessed to the browser. Any valid ESI elements can be included inside the block. It is not possible to nest **<!--esi -->** tags inside of each other.

```
<!--esi
ESI elements
-->
```

Example:

```
<!--esi
<esi:include src="http://www.akamai.com/ad.html"/>
-->
```

With the **<!--esi -->** tags, if the page is not processed, the whole block is simply hidden from the user.

Inserting Plain Text

The **text** statement allows you to insert flat text—any characters whatsoever—without interpretation or encoding by ESI. This provides a easy way to avoid errors caused by inadvertently including an unescaped dollar sign ($) or other reserved characters.

text

The **text** statement has a long form only.

```
<esi:text>
  This text can include dollar signs $, quotes "'" or any other flat
      text, and it will not be interpreted or encoded by ESI.
</esi:text>
```

This statement can be nested inside other ESI statements. If other ESI statements are nested inside of **esi:text**, however, they are considered to be flat text and not processed by ESI.

For other methods of avoiding encoding, see "Escaping the $ and Other Reserved Characters" on page 59.

Placing Variables and Functions Outside ESI Blocks

The **vars** tag allows you to use environment variables or functions outside of ESI blocks—inside an HTML code line, for example, instead of inside an **include** statement or **choose** block. Variables and functions are described in subsequent chapters in this document.

If you place the variable or function inside an ESI block, for example as part of an include statement, you do not use this construction.

vars

The **vars** tag has a short form and a long form.

Long form:

```
<esi:vars>Put some HTML and variable here</esi:vars>
```

Example of a variable:

```
<esi:vars>
<img src="http://a.b.com/$(HTTP_COOKIE{'type'})/hello.gif"/>
</esi:vars>
```

Example of a function:

```
<esi:vars>
The current time is $http_time($time())
</esi:vars>
```

Functions are described beginning on page 67. This construction would return the current time in the standard HTTP format.

Other **<esi:vars>** examples are found throughout this document.

Short form[1]**:**

```
<esi:vars name="VARIABLE_NAME"/>
```

This form can be used with standalone variables. For example, to report a variable in a table cell:

```
<tr>
<td>
<esi:vars name="$(GEO{'city'})"/>
</td>
```

Do Not Nest vars Inside an HTML Tag

The **esi:vars** tag cannot be nested inside an HTML code line. The following is an example of ***incorrect*** syntax:

```
<img src="http://abc.com/<esi:vars>$(HTTP_COOKIE{'type'})</esi:vars>/
      hello.gif"/>
```

1. The short form of **vars** is in addition to ESI 1.0 specifications. See "The ESI Specification" on page 8.

Explicit Exception Handling

The ESI language provides an explicit exception handling construct, the **try** block.

try | attempt | except

The **try** block is formulated as follows:

```
 <esi:try>
      <esi:attempt>
          Try this...
      </esi:attempt>
      <esi:except>
          If that fails, do this...
      </esi:except>
</esi:try>
```

Example:

```
<esi:try>
      <esi:attempt>
          <esi:comment text="Include an ad page"/>
          <esi:include src="http://www.akamai.com/ad1.html"/>
      </esi:attempt>
      <esi:except>
          <esi:comment text="Just write some HTML instead"/>
          <a href=www.akamai.com>www.akamai.com</a>
      </esi:except>
</esi:try>
```

The **except** statement is optional. EdgeSuite first processes the **attempt** sub-block. A failed ESI **include** statement triggers an error; the processor then attempts to execute the contents of the **except** sub-block, if one is present. Statements other than **include** do not trigger this error. The **except** sub-block is *not* triggered if the **src** object, **alt** object, or a default object is used, if an HTTP 200, 301, 302, and 401 response is received, or if an **onerror="continue"** attribute is applied (see the discussion below). In versions prior to 4.8, response codes other than 200 would trigger a response, and you can choose to configure for that behavior.

In the example above, if the ad page cannot be fetched, include a simple link instead. As is the case with the **choose** construct, valid statements in the **try** block must be placed inside an **attempt** or **except** sub-block. In the following example, the lines marked with daggers (†) are invalid and are discarded by the EdgeSuite processor.

```
<esi:try>
      † Invalid HTML here
      <esi:attempt>
          <esi:include...>
          This line is valid and will be processed.
      </esi:attempt>
      † Invalid HTML here
      <esi:except>
          This HTML line is valid and will be processed.
      </esi:except>
      † Invalid HTML here
</esi:try>
```

The very nature of the **try** block means that the processing behaviors of certain **esi:include** attributes are different from the behavior in other ESI formations. Two noteworthy cases are the **onerror="continue"** and the **maxwait** attribute.

Maxwait in a try Block

Take the case in which a **maxwait** is specified in the **try** block statement and the origin returns an HTTP 500 error before the **maxwait** expires. If EdgeSuite has a stale (TTL-expired) object available and the configuration is to serve the stale fragment if a new one can't be fetched, then EdgeSuite serves the stale fragment within the **esi:try**. This could be expected as EdgeSuite standard behavior.

However, in the same situation, if the **maxwait** were to expire, then EdgeSuite would *not* serve the stale content. Instead the processing would go on to the next **try** block action (another **esi:attempt** or an **esi:except**, for example), or to another configured failure action set to occur when ESI returns an error instead of the object.

Onerror in a try block

If you use the **onerror="continue"** attribute in the **include** statement inside the **try** block, you run the risk of defeating the block's purpose. If the **attempt** fails and the **onerror** attribute tells EdgeSuite to skip the **include**, the **except** block will not be processed and used for that **include**. One situation in which you may want to use the onerror attribute is when you use multiple **include**s inside the try block, and you do not want a failure in specific **include**s to trigger the **except** block.

esi:assign in a try Block

Also, before using **assign** in a **try** block, note the discussion on page 52.

Comments

You can add comments to a document using the **comment** tag.

comment

The **comment** tag is formulated as follows:

```
<esi:comment text="text commentary"/>
```

Example:

```
<esi:comment text="the following animation will have a 24 hr TTL."/>
```

These comments are for your own use, are not processed, and are simply deleted by EdgeSuite when the file is processed. They are not included in the final output.

CHAPTER 6. ESI Variables Support

This chapter discusses the three types of environmental variables supported in ESI:

- HTTP response and request headers
- Akamai-specific environment variables
- User-defined variables

In general, environment variables are automatically passed from templates to fragments—from parent to child—but cannot go the other direction. An exception to this general rule is the **`eval`** statement, described on page 25.

You can use variables as a part of ESI statements, or in a standalone manner—the **`vars`** statement (page 41) shows how you might use a standalone variable in a table cell. You can use variables in an **`include`** statement, but you cannot use a compound expression (Table 5 on page 63) as part of an **`src`** or **`alt`** object. That is, if you want to use a compound expression as part of an object, first define it as a variable, then include the variable in the statement.

HTTP and Other Client Headers

There are some limitations and requirements on honoring HTTP and other client headers in EdgeSuite. Some headers are not honored and some require enabling in the EdgeSuite configuration file in order to be used. For more details, see the document, *EdgeSuite Handling of Edge-Control and Other HTTP Headers*.

In EdgeSuite generally, within the limitations and requirements, HTTP headers are honored in requests from clients and in responses from the origin. Most relevant here, the HTTP headers can be used as variables, with the following conversion structure:

- The term “HTTP_” is prefixed to the name.
- The variable must be in upper case.
- Dashes (-) are replaced with underscores (_).

For example, **`Accept-encoding`** becomes **`HTTP_ACCEPT_ENCODING`**, **`Cookie`** becomes **`HTTP_COOKIE`**, and **`Host`** becomes **`HTTP_HOST`**.

The format of the variable reference is $(VARIABLE_NAME).

For example, to return the value of the HTTP_HOST variable:

```
$(HTTP_HOST)
```

Or, an example used in an ESI statement, in this case a Boolean expression:

```
<esi:when test="$(HTTP_COOKIE{'group'})=='Advanced'">
```

This same conversion structure can be used on other client headers of the type, **`HeaderName: Value`**. For example, the value in the **`Remote_User`** header may be accessed as **`$(HTTP_REMOTE_USER)`**.

Cookie Support

There are a variety of considerations with regard to cookie handling in ESI, because of caching considerations (not caching objects associated with unique cookies), cookie-only session IDs, and because on any one browser page there may be several objects, templates and fragments, that might be associated with cookies. Note that:

- Besides supporting the cookie header as an HTTP header, ESI provides a variable substructure to return cookie data. See Table 4 on page 50.
- ESI also provides a function to create and add a Set-cookie header. See the **`$add_header()`** function on page 74.
- Cookies can be used in expressions to define variables in the **`esi:assign`** statement, described on page 51.
- Cookies set on fragments as well as templates will be collected and sent to the end user's browser.
- Cookies set on the template can be propagated to fragments, if this is enabled in your EdgeSuite configuration. In propagation, cookies are validated for expiration time, security, and domain, including international domains (uk, ch, etc.).
  - Expired cookies of the same name are invalidated, and more current data from the root response header replaces the old data.
  - Secure cookies are propagated when the root template uses HTTPS.
- With the above enhancements, you can use Cookie-based session IDs in ESI. If you do so, talk with your Akamai technical representative, because specific setup in your EdgeSuite configuration file is required. Also, the configuration on your ESI templates and fragments should be set to "no-store;" a non-caching connection with the origin is needed to utilize cookie-based session IDs.

For more information on EdgeSuite Cookie Handling, see the *EdgeSuite Configuration Guide*. For information specifically on cookies and Session IDs, see the document, *EdgeSuite Session ID Support*.

POST Support

Origin servers can respond to client POST[1] requests through ESI, providing that ability is enabled in your EdgeSuite configuration file. POST processing is enabled by default but can be disabled should you choose to do so.

The values contained in the POST are available in the ESI document as a dictionary under the name POST, thus making the POST data available under the variable call,

1. This POST handling is in addition to ESI 1.0 specifications. See "The ESI Specification" on page 8.

$(POST{'sub1'}), where **sub1** is the name key in a name:value pair. The ESI dictionary expression is defined in Table 5 on page 63. See also an example of POST usage on page 36.

Currently, ESI only accepts the content-type "application/x-www-form-encoded," the standard POST mechanism in all browsers. ESI does not accept mime encoded POST content, which is traditionally used only for file upload. ESI accepts a maximum of 16384 bytes in POST responses; larger responses will generate an HTTP 500 error.

Akamai-Specific Variables

Table 3: Akamai-Specific Variables Supported in ESI

VARIABLE NAME	TYPE	EXAMPLES
GEO (see "The GEO Variable," below, and see Table 4)	strings returned from several keys	Country code: US Region code: CA Network type: dsl
QUERY_STRING	a collection of name=value pairs, pairs separated by a "&"	'first'='Robin'&'last'='Roberts'
REMOTE_ADDR	end user's IP address	123.234.243.132
REQUEST_METHOD	the HTTP request method	GET, POST
REQUEST_PATH	the path specified with the HTTP request method	/esi/example.html
Custom defined for values extracted from cookies or other elements. (see discussion next page)	strings	SessionID{'value'}
TRAFFIC_INFO (Bandwidth usage variable—see discussion below)	A dictionary (see page 63) of bandwidth usage values.	$(TRAFFIC_INFO{BW})

Variable names must be in UPPER CASE. The format of the variable reference is $(VARIABLE_NAME). For example:

```
$(REMOTE_ADDR)
```

Example of usage in an **include** statement:

```
<esi:include src="http://search.akamai.com/
      search?query=$(QUERY_STRING{'query'})"/>
```

The GEO Variable

GEO[1] data is provided by Akamai's EdgeScape and EdgeScape Pro services. Accessing GEO key values requires enabling an option in your EdgeSuite Configuration. For a complete explanation of GEO, its keys and potential results, see the *EdgeScape and EdgeScape Pro Users Guide*, in particular the chapter, "Implementing the EdgeScape Pro Service APIs," and the appendices.

1. The **GEO** variable is in addition to ESI 1.0 specifications. See "The ESI Specification" on page 8.

Note that you cannot use an EdgeScape default key or value in ESI, but you can set a default value on a GEO, as you can on any other variables, in ESI. Setting default values is described on page 56.

Also, the **Domain** and **Company** data available from EdgeScape Pro are not available in ESI.

TRAFFIC_INFO: Bandwidth Usage Variables

With the Usage Control EdgeSuite feature, you can define a variable in EdgeSuite configuration representing the bandwidth used in serving content to end users. The variables are define by:

- a CP Code or range of CP Codes (the CP Code is a unique number or set of numbers associated with your Akamai account contract.)
- a name.
- for the CP Code or CP Code range,
- For each variable, one of the following:
  - the current *bit* usage per second for streaming and other traffic.
  - the total *megabyte* usage since the beginning of the current calendar month for streaming traffic.
  - a fixed number (for example, to represent a threshold level).

The current bits-per-second may have a lag time of up to ten minutes from the time the data is sampled and propagated to the time it is available at the edge servers. The megabyte usage since the beginning of the month is updated every six hours.

You can have more than one variable defined and accessed. You can then access the variable in ESI as a name key of the dictionary, **`TRAFFIC_INFO`**.

Note, however, that this variable is created as a floating point number, but, like other variables passed to ESI, is accessed as a string in ESI. If you want to do numeric comparisons, you can do a rough conversion of the variable to an integer by truncating the period (.) and everything after it, then converting the remaining integer-like string to an integer, using ESI functions as follows. In this example, the key passed to ESI is BW1.

```
$int($substr($(TRAFFIC_INFO{BW1}), 0, $index($(TRAFFIC_INFO{BW1}), '.')))
```

You could then use this integer in evaluations. For example:

```
<esi:assign name="NEW_BW" value="$int($substr($(TRAFFIC_INFO{BW1}), 0,
      $index($(TRAFFIC_INFO{BW1}), '.')))"/>
<esi:choose>
  <esi:when test="$int($(NEW_BW))>2000">
      $set_redirect('lightestweightfile.html')
  </esi:when>
  <esi:when test="($int($(NEW_BW))>1000)&&($int($(NEW_BW))<2000)">
            <esi:include src="secondary-file.html"/>
  </esi:when>
  <esi:otherwise>
      <esi:include src="primary_fragment.html"/>
  </esi:otherwise>
</esi:choose>
```

This construction says to redirect the entire request to the lightest weight alternative if NEW_BW is over 2000; if NEW_BW is over 1000 but less than 2000, include a fragment that is lighter than the original but not the primary. Otherwise, use the primary fragment.

One thing you would want to add to this example is an **`$exist()`** test to check that **`$(TRAFFIC_INFO{BW1})`** is not null—that is, that the key has successfully been passed and accessed in ESI before attempting to use it.

Extracted Values

You can extract values from cookies, path components or parameters, or query strings, and pass them to ESI as variables. This extraction is discussed in the *EdgeSuite Configuration Guide* and in the *EdgeSuite Session ID Support* guide.

Substructures

Variable substructures can be accessed by key with a **`$(VARIABLE{key})`** structure. In the following example, the variable value could be set to override the existing value for the query. For example:

```
<esi:include
  src="http://search.akamai.com/
      search?query=$(QUERY_STRING{'query'})"/>
```

Substructure access makes sense for certain variables only. Key-based access makes sense for all variables consisting of logical key-value pairs. Table 4 on page 50 lists examples of substructure access for those variables that accept them, with specific values based on the examples in Table 3. Note that ESI supports the virtual keys, **`browser`**, **`version`**, and **`os`**, are accepted on HTTP_USER_AGENT.

Substructure Example

This example uses the **`<esi:vars>`** tag, described on page 41.

When requested from Internet Explorer, the following ESI markup

```
<esi:vars><img src="http://www.xyz.com/oop.cgi?browser=
      $(HTTP_USER_AGENT{'browser'})"></esi:vars>
```

…resolves to this in the output HTML:

```
<img src="http://www.xyz.com/oop.cgi?browser=MSIE">
```

Table 4: Variable Substructure Access

Example Access	Description / Example Values
$(GEO{*key*}) where the key can be one of the elements listed in the right column. For example, $(GEO{'country_code'}). The values are all reported as strings. Note that the Domain and Company keys are not available in ESI. $(GEO)	continent = NA country_code = US region_code = CA city = SANFRANCISCO dma = 807 msa = 127 pmsa = 7360 areacode = 415+650 lat = 37.7753 long = 122.4186, county = SANFRANCISCO+SANMATEO fips = 06075+06081, timezone = PST network = ibm network_type = dsl asnum = 5413 zip = 60170 + 60240 - 60245 (The dash (-) indicates a range.) throughput=high (Other keys may be added as EdgeScape expands its capability.) The result returns values for all above keys.
$(HTTP_ACCEPT{'text/html'})	Returns 'text/html' if client browser reported 'text/html' as supported medium.
$(HTTP_ACCEPT_CHARSET{'iso-8859-5'})	Returns 'iso-8859-5' if client browser reported 'iso-8859-5' as supported charset.
$(HTTP_ACCEPT_ENCODING{'gzip'})	Returns 'gzip' if client browser reported 'gzip' as supported language.
$(HTTP_ACCEPT_LANGUAGE{'fr'})	Returns 'fr' if client browser reported 'fr' as supported language.
$(HTTP_CONTENT_LENGTH)	This does not return a string, but returns the length of the content entity, in bytes, e.g, 7512
$(HTTP_COOKIE{'visits'}) $(HTTP_COOKIE)	42 Returns a semi-colon-separated list of key=value pairs, e.g., id=571; visits=42
$(HTTP_USER_AGENT{'os'}) $(HTTP_USER_AGENT{'version'}) $(HTTP_USER_AGENT)	Win (Returns "WIN", "MAC", "UNIX" or "OTHER"). "UNIX" is also returned for Linux, SunOS and Solaris. 5.0.1 (Returns browser version on MSIE or Mozilla) The User-Agent string returned from the browser, as a semi-colon-separated list of key=value pairs, e.g., Mozilla 4.0; MSIE 5.0.1; WIN
$(QUERY_STRING{'last'})	Roberts

Table 4: Variable Substructure Access

$(TRAFFIC_INFO{BW})	8737.9933 (a string that looks like a floating point, representing bandwidth usage over some period).

Setting and Using User-Defined Variables—the assign Statement

The **esi:assign** statement provides for setting variables multiple types, including strings and compound expressions involving multiple types and operators described in "Expressions and Operations" on page 59. The **esi:set** statement is now deprecated, since it has been replaced by the more functional **esi:assign**. Code using **esi:set** will be supported for now but may be dropped in the future.

Regex Match Results as Variables

One type of variable assignation does not require either a **set** or **assign** statement, and that is when you create a variable from the results of a regex match. This is described on page 61.

assign

The **assign**[1] statement uses a short or long form to define a custom name and value.

Syntax

Short Form:

```
<esi:assign name="var_name{key}" value="value"/>
```

For example, using an integer:

```
  <esi:assign name="A_Number" value="123"/>
```

The **name** is composed of up to 256 alphanumeric characters (A-z, 0-9), and can include underscores (_) but cannot include a $ (dollar sign), which is reserved. The first character must be an alpha character of either case.

The **value** is an expression or compound expression—multiple expressions connected by mathematical, string, or Boolean operators, which are discussed in "Expressions and Operations" on page 59. The compound expressions can mix types, but The value cannot be empty or null—that is, **value=""** is not legal.

The {**key**} is optional—it is the index value for a list or the name key for a dictionary. Dictionary and list subkeys in **assign** are discussed on page 53.

In the short form, the value is surrounded by double quotes. In the long form, the value is whatever is inside the opening and closing phrases, and the type of the expression is identified by conventions described on page 63. You can use the backslash (\) (page 59) in the **assign** statement to include characters you could not otherwise include.

Long Form:

```
<esi:assign name="name">
      'This is the value, an expression consisting of 1 or more types
      and operators.'
</esi:assign>
```

1. The **assign** tag is in addition to ESI 1.0 specifications. See "The ESI Specification" on page 8.

Example:

```
<esi:assign name="Favorites">
      {'Sports': 'football, basketball, soccer', 'Music': 'classical
      jazz, fusion, Latin jazz', 'Books': 'popular fiction, Sci-fi'}
</esi:assign>
```

Usage

You can use **assign** as an ESI statement by itself or within an ESI block such as the **choose** block. The value is passed from template pages to fragments, but not from fragments to templates. The value is recalled with **$(*name*)**. For example:

```
<esi:assign name="foo" value="12345"/>
<esi:choose>
<esi:when test="$(foo)==12345">
    <esi:assign name="foo" value="54321"/>
  </esi:when>
</esi:choose>
<esi:vars>
This should print out "54321": $(foo)
</esi:vars>
```

This example creates the **foo** variable with an initial numeric value of **12345**. The **when test** evaluates whether **foo** is equal to **12345**. Since it is, the next line reassigns **foo** to **54321**. The choose block is closed, and the next line recalls the value of **foo**, which is now **54321**.The purpose of the **<esi:vars>** tag is further described on page 41; in brief, it allows the ESI-assign variable **foo** to be used outside any ESI statement or block—in this case, in the "This should print…" line.

Other examples of esi:assign can be found throughout this document, and particularly in the discussions of functions, beginning on page 67.

Usage in a try Block

Because of the way the pages and blocks are processed, the values of user-defined variables assigned or reassigned inside a **try** block cannot be passed to statements outside the block or to other statements inside the block; they can be used only in the specific statement and its children. To illustrate this point, look at this code logic:

```
  <esi:try>
      <esi:attempt>
          assign Variable 1
          include Object 1 using Variable 1
      </esi:attempt>
      <esi:attempt>
          assign Variable 2
          include Object 2 using Variable 2
      </esi:attempt>
      <esi:attempt>
          assign Variable 3
          include Object 3 using Variable 3
      </esi:attempt>
</esi:try>
```

In this case, none of the variables are used outside the **try** block, except that *Variable 1* would be used with *Object 1* and its children if that **attempt** completes, *Variable 2* would be used with *Object 2* and its children if that **attempt** completes, etc.

An Exception to the try Block Rule—the eval Statement

An exception to the above logic occurs when variables are included using the eval statement, described on page 25.

Lists, Dictionaries, and Subkeys in the assign Statement

Dictionaries and lists are defined in Table 5 on page 63. You can assign subkeys in lists and dictionaries using the following construction:

```
<esi:assign name="list_or_dict{key}" value="somevalue"/>
```

- Lists keys are index numbers; the index key is a integer indicating the item's position, beginning with 0 for the first position.
- For dictionaries, the key is the name in a name:value pair.

A List Example

In the following example, the first line creates the list, **colors**, and assigns to it three members, **red**, **blue**, and **green**, representing positions 0, 1, and 2. The second line replaces **red** with **purple** at position 0.

```
<esi:assign name="colors" value="[ 'red', 'blue', 'green' ]"/>
<esi:assign name="colors{0}" value="purple"/>
```

Printing out **colors**, the output is **['purple', 'blue', 'green']**.

A Dictionary Example

In this next example, the first line creates the dictionary, **ages**, and assigns entries. The second line changes the value to **28** for dictionary name key, **joan**.

```
<esi:assign name="ages" value="{ 'bob' : 34, 'joan' : 27, 'ed' : 23 }"/>
<esi:assign name="ages{joan}" value="28"/>
```

Here's an alternate construction for the second line, taking advantage of the fact that the dictionary value is an integer:

```
<esi:assign name="ages{joan}" value="$(ages{joan}) + 1"/>
```

Either way, the dictionary output is **{'bob':34, 'joan':28, 'ed':23}**

List and Dictionary Subkey Guidelines

Using the **assign** statement subkey syntax, the following guidelines apply:

- You can create *dictionaries*, assign name keys and values on the fly, but *lists* must already exist with all their members when you use the subkey construction. That is, you cannot assign to an entry to a list index if the index position doesn't exist. On the other hand, if you assign an entry to a dictionary name key that doesn't exist, you'll append a new name : value pair to the dictionary.
- You can refer to at most one subkey on "name" side of the **assign** statement.
- Lists and dictionaries are assigned by reference, not by copy.

The subsequent discussion is of examples and consequences of these guidelines.

Dictionaries Keys Can Be Created on the Fly, but Lists Must Exist

For dictionaries, you can create new keys on the fly. Lists, however, are limited to the creation size. The different behaviors can result in significantly different results. Take, for example:

```
<esi:assign name="ages" value="{ 'bob' : 34, 'joan' : 27, 'ed' : 23 }"/>
<esi:assign name="colors" value="[ 'red', 'blue', 'green' ]"/>
```

The variable **ages** is dictionary, while **colors** is a list, and assigning keys (for dictionaries) or indexes (for lists) have different results.

We can append a new name : value pair to the dictionary as follows:

```
<esi:assign name="ages{ronald}" value="56"/>
<esi:vars name="ages"/>
```

Output:

```
{ 'bob' : 34, 'joan' : 28, 'ed' : 23, 'ronald' : 56 }
```

But if we attempt to add a fourth list member to the three-member list, **colors**, we get a 500 error.

```
<esi:assign name="colors{3}" value="yellow"/>  Error!
```

As an aside, note that list indexes are always integers. The following statement results in an error because it attempts to assign a string name key where only an integer would be accepted:

```
<esi:assign name="colors{joe}" value="black"/>
```

Not only can you create new dictionary keys on the fly, you can also create new dictionaries. A dictionary does not need to exist before its subkey is assigned to, but a list must. If a variable does not exist and a numbered index key is assigned, ESI creates a dictionary instead, using the string representation of the index numbers as the name key. To illustrate: even though you might intend creation of a list, this code creates a dictionary with one pair, **0 : yellow**; note that "0" is a string.

```
<esi:assign name="newlist{0}" value="yellow"/>
```

Remember that when you iterate over a dictionary, no order is guaranteed (see the discussion on page 34). There may be times you need to create a indexed list. To do this you can first create an empty list of the size you need; you can then modify the list member by referring to its position. For example:

```
<esi:comment value="Create a list of size 4"/>
<esi:assign name="newlist" value="[ 0, 0, 0, 0]"/>
<esi:assign name="newlist{0}" value="yellow"/>
```

Refer to Only One Subkey on the Name Side of the assign Statement

You can refer to at most one subkey on the name side of the assignment:. The second line below results in an error because it attempts to refer to more than one subkey.

```
<esi:assign name="complex" value="['one',[ 'a', 'b' 'c'],'three' ]"/>
<esi:assign name="complex{1}{1}" value="x"/> ERROR!
```

To accomplish the result without generating an error:

```
<esi:assign name="list_in_complex" value="$(complex{1})"/>
<esi:assign name="list_in_complex{1} value="x"/>
```

The statement **<esi:vars name="complex"/>** now outputs:

```
[ 'one', [ 'a', 'x' 'c' ], 'three' ]
```

Note the successful replacement of "x" with "b" in the second position (position 1).

Lists and Dictionaries are Referenced, Not Copied

The previous examples works in the way illustrated because lists and dictionaries are assigned by reference rather than by copy—a fact that can give you unexpected behavior if you expect the variable to be copied. For example:

```
<esi:assign name="list" value="[1, 2, 3]"/>
<esi:assign name="copy1" value="list"/>  Does not copy!
<esi:assign name="copy2" value="list"/>  Does not copy!
<esi:assign name="copy1{2} value="9"/>
```

The output would be as follows:

```
<esi:vars>
$(list)
$(copy1)
$(copy2)
</esi:vars>
```

Output:

```
[1, 2, 9]
[1, 2, 9]
[1, 2, 9]
```

The three output lines are the same because they all refer to the same list. Changing one list changes them all, since they are assigned by reference, not by copy. This referencing process applies to dictionaries as well as lists.

To achieve results that you would get by copying, you need to iterate over the original structure and make assignments key by key. This iteration example uses **foreach**, discussed on page 34):

```
<esi:assign name="dict" value="{1 : 'one', 2 : 'two', 3 : 'three'}"/>
<esi:foreach collection="dict"/>
<esi:assign name="copy{$(item{0})}" value="$(item{1})"/>
</esi:foreach>
<esi:assign name="copy{2}" value="Second"/>
```

The iteration has created a copy of the dictionary. If you now output both, you can note the change in the copied version; it has been modified as needed:

```
<esi:vars name="dict"/>
<esi:vars name="copy"/>
```

Output:

```
{ 1 : 'one', 2 : 'two', 3 : 'three' }
{ 1 : 'one', 2 : 'Second', 3 : 'three' }
```

This iterative operation can be more difficult to accomplish for lists, since you must first create the copy list with the correct size before you perform the copy. When you do not know the size of the list beforehand, this proves to be difficult but possible with the use of the **eval** statement (page 25).

Setting Defaults

Attempted access to undefined variables, empty variables, or non-existent sub-structure (unknown key name) evaluates to **null** (empty). In cases where evaluation can yield a null value, you can specify the substitution of a default value instead, using the following syntax where *default* is an expression. See Table 5 on page 63 for details on expressions in ESI.

```
$(VARIABLE|default)
```

For example:

```
<esi:include
      src="http://www.xyz.com/$(HTTP_COOKIE{'cobrand'}|'akamai').htm"/>
```

will result in EdgeSuite fetching the following URL if it cannot evaluate "cobrand" from the cookie.

```
http://www.xyz.com/akamai.html
```

Another example: this construction sets "en-us" as the default language for the HTTP_ACCEPT_LANGUAGE variable.

```
<esi:vars><IFRAME src="$(HTTP_ACCEPT_LANGUAGE{'en-gb'}|'en-us')/
      flag.htm"></esi:vars>
      -- alternative HTML markup for IFRAME --
</IFRAME>
```

Using Expressions Other Than Strings

Expressions other than strings can be used as defaults. Here's an example of a user-defined variable used as a default value. This sets a user-defined variable named "Default_Browser," and then uses its value as the default user agent. (See page 51 for more information about the **esi:assign** statement.)

```
<esi:assign name="Default_Browser"> 'OTHER'</esi:assign>
<esi:vars>
<img src="http://www.xyz.com/oop.cgi?browser=
      $(HTTP_USER_AGENT{'browser'} | $(Default_Browser))">
</esi:vars>
```

If $(HTTP_USER_AGENT) cannot be evaluated, the value defaults to $(Default_Browser), which in this case evaluates to "OTHER."

Testing for an Empty or Non-Existent Variable

You can use the ESI functions, **$is_empty()** and **$exists()**, to check for empty or nonexistent variables, as described on page 74.

You can also use the following methods inside a **choose** block:

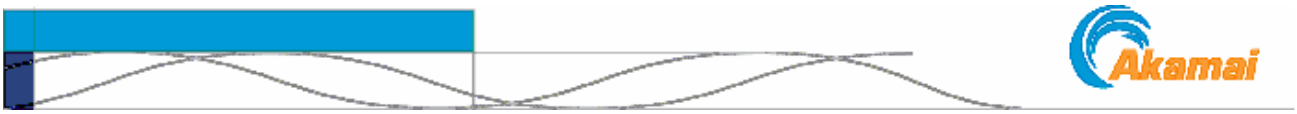
```
<esi:when test="$(foo)">
```

If the variable is empty or non-existent, the test evaluates to “false.” To return “true” if the variable is empty or non-existent, use the following construction:

```
<esi:when test="!$(foo)">
```

The following would test for the existence of a query string:

```
<esi:when test="$(QUERY_STRING)">
```



CHAPTER 7. Expressions and Operations

This chapter covers the boolean operations, logical and string operations, expressions, and escape operations used in EdgeSuite ESI.

Escaping the $ and Other Reserved Characters

Certain characters are reserved in ESI. For example, the $ indicates a function or variable evaluation will follow, and the single quote delineates the beginning or ending of a string. You can use the **<esi:text>** statement to insert pure text, as described on page 40, or you can place any character as a literal in any ESI code by preceding it with a backslash (\)[1]. To use a user-defined variable as an example:

```
<esi:assign name="pitch" value="'You\'ll get amazing products.'"/>
```

This would yield the value, "You'll get amazing products." You can use a double-backslash to yield a literal backslash. The following would yield "\Program Files\Game\Fun.exe."

```
<esi:assign name="game" value="'\\Program Files\\Game\\Fun.exe.'"/>
```

Triple Single Quotes

The backslash works inside any ESI code or block of code, including **esi:include** statements and attributes. You can also use triple quotes—pairs of three single quotes—to surround the strings, instead of single quotes, to indicate literals, including the backslash and the dollar sign ($). This is described in Table 5 on page 63. For example, in previous versions of ESI, you could have used the following statement to yield the value, "\Program Files\Games\Fun.exe":

```
<esi:assign name="game" value="'\Program Files\Game\Fun.exe.'"/>
```

Beware of Quadruple Single Quotes

Using the triple quotes, this statement yields the same value:

```
<esi:assign name="game" value="'''\Program Files\Games\Fun.exe.'''"/>
```

A Caveat and a Solution

Note: When using triple-single quotes, do not place them next to another single quote—that is, *do not use four single quotes. The following would yield an error*:

```
<esi:assign name="quote" value="''''It's the 'Cat's Meow.''''"/>
```

A solution is to add a space after the first of the four single quotes. In the above example, the final phrase would become **'Cat's Meow.' ''''/>**

1. The \ operator and the triple quote string function are extensions to the ESI 1.0 specifications. See "The ESI Specification" on page 8.

Boolean Expressions

A Boolean expression, shown in this document as **BOOL-expr**, evaluates to logical true (value=1) or false (value=0). These operators are most frequently used with the **when** statement in the **choose** block, discussed on page 31. The following set of unary and binary logical operators are supported, listed in order of decreasing precedence:

OPERATOR	TYPE
==, !=, <, >, <=, >=, has, has_i, matches, matches_i	comparison
!	unary negation
&& (or & when bitwise is not used)	logical and
|| (or | when bitwise is not used)	logical or

If you choose to bitwise operators as described on page 64, you must use **&&** and **||** in place of **&** and **|**, respectively. The bitwise behavior is off by default but may be enabled in your EdgeSuite configuration file. It is highly recommended to convert documents using & and | as logical operators to the new format.

Note that the use of **&** is query strings is not affected by this, since in query strings the **&** is not a logical AND, but is a separator by convention.

The follows rules apply to the Boolean operations:

- Operands associate from left to right. Sub-expressions can be grouped with parentheses to explicitly specify association.
- If both operands are numeric, the expression is evaluated numerically. If either binary operand is non-numeric, both operands are evaluated lexicographically as strings. Values returned for environmental variables may be evaluated as numbers, but care should be taken. For example, a version reported as 3.01.23 or 1.05a will not test as a number.
- Single or triple (three single) quotes must be used to delimit string literals.

```
$(HTTP_COOKIE{'first_name'}|'new user')
```

- If any operand is empty or undefined, the expression is evaluated to be false.
- The logical operators (&&, !, ||) are used to qualify comparisons.

Examples of correct syntax:

```
!(1==1)
!('a'<='c')
(1==1)|('abc'=='def')
(4!=5)&&(4==5)
```

has and has_i

The **has** and **has_i**[1] operators check to see if an expression contains a particular string; **has_i** performs a case insensitive evaluation. These operators are useful in

1. The **has_i** operator is an extension to the ESI 1.0 specifications. See "The ESI Specification" on page 8.

checking for strings in the environment variables discussed in the next chapter. The comparison can be used on a variable or on a part of a variable—for example, a key, a value, or part of a key or value. For example, the following constructions are all valid:

```
<esi:when test="$(HTTP_COOKIE) has 'Sam'">
<esi:when test="$(HTTP_COOKIE{'first_name'}) has 'Sam'">
<esi:when test="$(HTTP_COOKIE{'first_name'}) has_i 'sam'">
```

In the first two examples, case matters: "Sam" is not the same as "sam."

- The first test, **`$(HTTP_COOKIE) has 'Sam'`**, evaluates to true if "Sam" is found anywhere in the result (e.g., first_name=Sam, last_name=Samuelson, or location=Samoa).
- The second test, **`$(HTTP_COOKIE{'first_name'}) has 'Sam'`** evaluates to true only if "Sam" is a result of the evaluation of the first_name key.
- The third test, the **`has_i`** evaluation, is similar to the second but evaluates to true if it finds a case variant of "Sam," such as "sam," "SAM," or "saM."

Regular Expression Evaluations

matches and matches_i

The **`matches`**[1] operator checks to see if an expression contains an *extended regular expression* (ERE) such as is used in the UNIX *grep* family; **`matches_i`** performs a case insensitive evaluation. Examples:

```
<esi:when test="$(HTTP_CONTENT_LENGTH) matches '''^[0-9]{5,}$'''">
<esi:when test="$(HTTP_USER_AGENT{'version'}) matches '''4\.[0-9]+'''">
<esi:when test="$(HTTP_USER_AGENT{'os'}) matches_i '''^u'''">
```

The first test would match any CONTENT_LENGTH evaluated to 5 or more digits—10000 or more bytes. The second test would match versions returned if they contained the term, 4.0, 4.56, 4.9a, etc. The third test would match "UNIX," "unix," or "Unix," for example.

You can use the case insensitive pattern matches modifier, "i" within a regular expression to match patterns in which some part of the expression needs to be case-sensitive and another part doesn't. For example:

```
<esi:when test="helloWORld matches '''hello((?i)world)'''">
```

evaluates to true, but...

```
<esi:when test="HelloWORld matches '''hello((?i)world)'''">
```

does not. The upper case "H" in the second example does not match.

Accessing Regex Results as Variables or Variable Subkeys

The matched expressions resulting from a **`matches`** or **`matches_i`** evaluation are automatically assigned to ESI variables. Putting parentheses in your regular expression further allows you to extract particular matched subexpressions into ESI variables.

---

1. The **`matches`** and **`matches_i`** operators are extensions to the ESI 1.0 specifications. See "The ESI Specification" on page 8.

By default, the matched expression and subexpressions are assigned to a variable called MATCHES, but you can assign them to another variable name by using the "matchname" attribute within your esi tag.

Within the ESI variable the indexes refer to each particular matched subexpression, with index 1referring to the subexpression matched in the first parentheses, index 2 to the second parentheses, etc. Index 0 is special and refers to the entire matched expression.

The matched subexpressions can then be accessed through the normal ESI method of accessing variables—that is, **`$(VARIABLE)`** or **`$(VARIABLE{key})`**. Variables and their subkeys are discussed on page 49.

Example of Matched Regular Expression Assignment to a Variable

```
<esi:assign name="myString" value="'xxx123foo456xxx'"/>
<esi:choose>
  <esi:when test="$(myString) matches  '''([0-9]+)([a-z]+)([0-9]+)'''"
      matchname="myValue">
          entire match is $(myValue{0})
          first subexpression is $(myValue{1})
          second subexpression is $(myValue{2})
          third subexpression is $(myValue{3})
  </esi:when>
</esi:choose>
```

This would print out:

```
entire match is 123foo456
first subexpression is 123
second subexpression is foo
third subexpression is 456
```

You can access the results of a regex by assigning the results to a variable, as follows:

```
<esi:when test="$(string) matches '''a([0-9]*)([a-Z]*)'''
      matchname='MYVAR'/>
```

The results are assigned to the variable "matchname." If you do not specify the matchname, a default, **`MATCHES`**, is used. The variable is accessed through the normal ESI method of accessing variables, for example, **`$(MYVAR)`** (or **`$(MATCHES)`** when the default is used). Subkeys are accessed as **`$(MYVAR{0})`**, **`$(MYVAR{1})`**, and so forth, where "0," "1," and other numbers are position keys: the first, second, and subsequent matches found.

Triple Single Quotes Recommended

Note that the examples above use the triple-quote (three single quotes) method, described in Table 5 on page 63, to surround the regular expression string. The triple quotes are recommended to allow for the use of the literal backslash (\) character in the expression. In ESI, the backslash would otherwise escape the character that follows it (as described on page 59). For more information on evaluations using regular expressions, see an extended regular expression reference, for example:

`http://www.opengroup.org/onlinepubs/7908799/xbd/re.html`

Expressions

This table lists the expressions you can use in ESI. The examples shown in this table use the **`assign`** statement (defines variables, described on page 51.)

Table 5: Expressions[a]

TYPE	DESCRIPTION & SYNTAX	EXAMPLES
String 1	Characters, other than the single quote surrounded by single quotes.	<esi:assign name="var" value="'Hello there, '"/> <esi:assign name="var">'Hello there, ' </esi:assign>
String 2	Any characters surrounded by triple single quotes. This allows use of literal single quotes, the dollar sign ($), or backslashes (\) in values. *Do not use four single quotes. This example would yield an error because of the four single quotes after "meow."*	<esi:assign name="var" value="'''He's using a literal \ and a $ in this form.''' "/> <esi:assign name="var">'''He's using a literal \ and a $ in this form.''' </esi:assign> *an illegal construction:* <esi:assign name="quote" value=''''It's the 'Cat's Meow.''''/> This can be made legal by adding a space after the first of the four single quotes: ''''It's the 'Cat's Meow.' ''''/>
Variable	Any legal ESI-supported variable.	<esi:assign name="var" value="$(HTTP_USER_AGENT{'os'})"/> <esi:assign name="var">$(HTTP_USER_AGENT{'os'})</esi:assign>
Function	An ESI function (see page 67).	<esi:assign name="var" value="$url_decode($(QUERY_STRING{'name'}))"/>
List	Enclosed with square braces, a list of items separated by commas. Lists can mix sub-types. Items are accessed using the position (starting at 0) as the index key.	<esi:assign name="var" value="[abc,def,efg,hij,klm,nop]"/> $(var{3}) would return "hij".
Dictionary	Enclosed with curly braces, a list of name: value pairs, separated by commas. Dictionaries can mix sub-types. Dictionary items are accessed using the pair value as the key.	<esi:assign name="var" value="{'sub1':'val1', 'sub2':123,'sub3':$(foo),'sub4':$(HTTP_USER_AGENT)}"/> $(var{'sub1'}) would return "val1"
Number	An integer between 2^{32}–1 and -2^{32}–1. (less than 2147483648 and greater than –2147483647). Outside these limits, an evaluation may fail or yield incorrect results.	<esi:assign name="var" value="123"/> or <esi:assign name="var" value="-123"/> An example of statement that will fail: <esi:assign name="num" value="2147483647+1"/>
Boolean expression	Uses a boolean operator (see p. 60)	<esi:assign name="var" value="1 == 2"/>
Compound expression	Uses a math or string operator, described below. These may not be used in an esi:include statement.	<esi:assign name="var" value="$(foo) + $(bar)"/>

a. Expressions other than the single quotes string are extensions to the ESI 1.0 specifications. See page 8.

Logical and String Operators

Table 6: Operators Used with Compound Expressions

OPERATORS	USED WITH
+ (add), - (subtract), * (multiply), / (divide), % (modulo—return the remainder)	Numbers
+ (concatenate)	Strings, Lists
* (repeat n times, e.g, 3*"string" returns "stringstring-string")	Strings, Lists
.. (returns a range, e.g. [1..5] returns 1,2,3,4,5)	Lists
Boolean operators (see page 60)	All

Bitwise Operations

Bitwise operators act on the internal binary representation of a number.

Table 7: Bitwise Operators

OPERATOR	ACTION
<<	Shift left (shifts the bits of a number left)
>>	Shift right (shiftf the bits of a number right)
~	Bitwise not (negation) (flips the bits of a number)
&	And ('ands' the bits of two numbers)
|	Or ('or' the bits of two numbers)
^	Xor ('exclusive ors' the bits of two numbers)

Note that to use these bitmap operators you need to have your EdgeSuite configuration set up to enable them. The default behavior is for ESI to see **&** and | as logical operators.

Here are some examples:

Assume you have two decimal 8 bit numbers, 11 and 6. Their binary representations are as follows:

6 = 00000110

11 = 00001011

The bitwise operators would operate on these as follows:

6 << 2 = 00011000 = 24

11 << 1 = 00010110 = 22

6 >> 1 = 00000011 = 3

11 >> 1 = 00000101 = 5

Note that shifting a number one place left or right essentially multiples or divides the number by 2.

More examples:

~6 = 11111001 = 249

~11 = 11110100 = 244

6 & 11 calculates as follows:

```
  0110 =  6
& 1011 = 11
-----------
0010 =  2
```

6 | 11 calculates as follows:

```
 0110 =  6
| 1011 = 11
-----------
1111 = 15
```

6 ^ 11 calculates as follows:

```
  0110 =  6
^ 1011 = 11
-----------
1101 = 13
```

Range Operations

The range operator, two dots (..) returns a list of sequential values from the left value to the right value. If the left value is more than the right value, the range is ascending, otherwise it is descending. The operator can only be used in the context of list creation, inside list delineators []. Some examples:

```
<esi:assign name="list" value="[0..7]"/>
<esi:foreach collection="[1..10]"> or
<esi:foreach collection="[5..1]"> or
<esi:assign name="list" value="[0..3, 5, 7..9]"/>
```

The above four lines could be expressed as:

```
<esi:assign name="list" value="[0, 1, 2, 3, 4, 5, 6, 7]"/>
<esi:foreach collection="[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]">
<esi:foreach collection="[5, 4, 3, 2, 1]">
<esi:assign name="list" value="[0, 1, 2, 3, 5, 7, 8, 9]"/>
```

The left or right range member can be a variable. The following is valid when **`$(max)`** evaluates to an integer:

```
<esi:foreach collection="[10 .. $(max)]">
```

The range operator can be interleaved in a list assignment:

```
<esi:assign name="list" value="[0..5, 8, 9, 12..78 ]"/>
```

Treating Strings as Lists

Strings can be treated as lists. Note the following variable construction and the access of subkeys, for example:

```
<esi:assign name="a_string">'abcde'</esi:assign>
<esi:assign name="a_character">$(a_string{0})</esi:assign>
<esi:assign name="another_character">$(a_string{3})</esi:assign>
```

In this situation, the **a_string** variable is created as a string, but can be manipulated as a list: **a_character** returns "a," the first item (position 0) in the list, and **another_character** returns "d," the fourth item (position 3) in the list.

Mixing Types in Concatenation: an Implicit Coercion to Strings

When one side of the concatenation addition operator "+" is a string, ESI attempts to coerce the other side into a string as well, and then perform the concatenation. For example, the following concatenation, used in creating the variable, would return the value, "You have 12 dollars:"

```
<esi:assign name="somemoney" value="'You have ' + 12 + ' dollars'">
```

This can useful when constructing larger strings for objects such as unique cookies. For example, this creates a variable, mycookie, by concatenating various string and number values:

```
<esi:assign name="mycookie" value="'ID0='+ $time() + '.' + $rand()
      +';expires=Mon Apr 2314:37:13 2012; path=/;'"/>
<esi:vars>
$add_header( 'Set-Cookie', $(mycookie) )
</esi:vars>
```

A Caveat

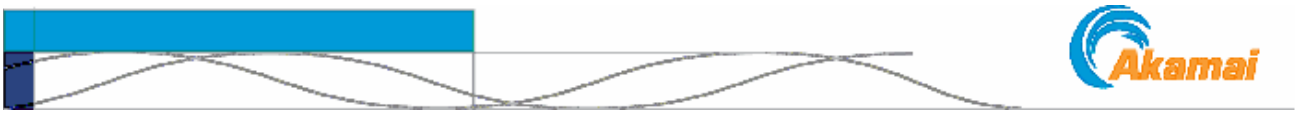
Remembering that ESI evaluates expressions left to right, the coercion will not occur in some situations. For example, the following formulation results in $(feb) evaluating to the string, "10 days in February:"

```
<esi:assign name="feb" value="2 + 8 + ' days in February'">
```

This is because the arithmetic addition, "2 + 8," is evaluated before the implicit coercion occurs in the concatenation. If you wanted "28 days" in February, you would need to explicitly cast the 2 or the 8 to a string. You can do this in a couple of different ways:

- Surround the two, or eight, or both, with single quotes, indicating a string: **value="'2' + '8'" +**
- Use the **$str()** function to cast one or both integers to strings, described on page 71: value=**"$str(2) + $str(8) + '**

It is not necessary to cast both numbers to a string because of the implicit conversion rules, but it is good to be explicit.



CHAPTER 8. ESI Functions

In EdgeSuite ESI the dollar sign, "$," has a special meaning: it indicates the beginning of a function call or a variable that can be used either inside or outside an ESI statement or block. The chapter, "ESI Variables Support," discussed a specific use of the $ to evaluate variables, but EdgeSuite ESI supports many other functions, including the following, which are discussed in this chapter:[1]

As is the case with variables, you can use these functions in any appropriate ESI statement, including **`include`** statements or **`choose`** blocks, and in assigning user-defined variables, as described on page 51.

The expressions and operations used in the examples in this chapter are defined in "Expressions and Operations" on page 59.

String Functions

ESI contains a set of functions, detailed in the body of this chapter, to specify string separators, change case, return substrings or index positions, strip white space, concatenate, or replace substrings.

Internationalization

`$convert_to_unicode()` and **`$convert_from_unicode()`** are used only with multibyte encoding, to convert multibyte text to and from ESI's internal encoding. These are discussed in the chapter on internationalization on page 88.

Literals

Note: you can also use the backslash(\) to embed literals in ESI code (see page 59), or the **`esi:text`** statement to include pure text (see page 40).

- **`$dollar()`** – returns a literal "$"
- **`$dquote()`** | **`$squote()`** – return literal double quote marks (") and single quote marks('), respectively.

String/Integer Conversion

- **`$str()`** casts integers to strings, and **`$int()`** casts strings to integers.

Inclusion of Binary Data

- **`$bin_int()`** allows for the insertion of binary data. For example, the following:

  ```
  <esi:vars>X$bin_int(127)X</esi:vars>
  ```

  Would output (in hex dump), **`58 7F 00 00 00 58`**

  The "58" is ASCII code (in hex) for "X" and "7F 00 00 00" is the little endian representation (in hex) of 127.

1. The function call capacity is in addition to ESI 1.0 specifications. See "The ESI Specification" on page 8.

Counting, Numbers, and Booleans

- **$len()** – returns the length of a string or sequence
- **$list_delitem()** – deletes a list item and returns the value of the deleted item.
- **$rand()** returns a random numbers, and **$last_rand()** returns the last number generated by **$rand()**.
- **$is_empty() | $exists()** – return a true (1) or false (0) if a variable or function is empty or if it exists, respectively.

HTTP Functions

- **$add_header()** – adds an HTTP response header.
- **$set_response_code()** sets an HTTP response code.
- **$set_redirect()** sets a redirect.
- **$add_cachebusting_header()** prevents caching in downstream (toward the client browser) servers or proxies.

Encode/Decode Functions

- **$url_encode() | $url_decode()** – encodes and decodes URLs with illegal (to HTTP) characters.
- **$html_encode() | $html_decode()** – encodes and decodes HTML code with illegal characters that need to be replaced

Time Functions

- **$time()** and **$http_time()**– returns time in Unix epoch and in standard HTTP format, respectively. **$strftime()**returns a specific formatted string.

String Functions

$string_split()

$string_split(string [,sep] [,max_sep]) allows you to split a string into a list using a specific character as a delimiter. It allows you to specify both a separator a maximum number of splits that will be executed, with the remainder left in the final segment. The ***string*** is the string to act on; **sep** is the optional character to use as a separator—the default is white space ' '; and **max_sep** is the optional maximum number of splits.

This can be used, for example, to assign the plus sign (+) or other character as the string separator to delimit stock quotes. Here's an example of an iteration over a query string using the plus sign (+) as the separator. No **max_sep** specified. The query string is **'stock1+stock2+stock3+stock4'**:

```
<esi:foreach item="stock_query"
      collection="$string_split($(QUERY_STRING),'+')">
$(stock_query)<br>
</esi:foreach>
```

And the result would be:

```
stock1
stock2
stock3
stock4
```

This function always returns a list. If the separator is not found in the string to be split, the list contains the entire string as the only item in the list.

$join()

`$join(words[, sep] )` takes a list of words and an optional separator and returns a string formed by the concatenation of the members of that list, separated by the separator. If no separator is specified, a single space is used by default. This function performs the exact opposite function of **`$string_split()`**.

In this example, the list shown first joined and separated by the default space, then by a comma, then a comma and space.

```
<esi:assign name="list" value="{ 'this', 'is', 'a', 'phrase' }"/>
<esi:assign name="str1" value="$join( $(list) )"/>
<esi:assign name="str2" value="$join( $(list), ',' )"/>
<esi:assign name="str2" value="$join( $(list), ', ' )"/>

$(str1) == "this is a phrase"
$(str2) == "this,is,a,phrase"
$(str3) == "this, is, a, phrase"
```

$index()

`$index(s,c)` returns the lowest index position, in 0-based indexing, in string **`s`** where the character **`c`** is found. Returns -1 if **`c`** is not found.

For example, **`$index('teststring','r')`** returns 6, since "r" is the 7th character.

$rindex()

`$rindex(s,c)` is similar to **`$index()`** except that it returns the highest index position. For example, **`$rindex('teststring','t')`** returns 5, the highest position of "t".

$lstrip()

`$lstrip(s)` returns a copy of string **`s`** but without leading white-space characters.

For example, **`$lstrip(" text")`** returns "text".

$rstrip()

`$rstrip(s)` returns a copy of string **`s`** but without trailing white-space characters.

For example, **`$rstrip('text ')`** returns "text".

$strip()

`$strip(s)` returns a copy of string **`s`** without leading or trailing white space.

For example, **`$strip(' text ')`** returns "text".

$replace()

`$replace(s, old, new[, maxsplit])` returns a copy of string **`s`** with all occurrences of substring **`old`** replaced by substring **`new`**. If the optional integer argument **`maxsplit`** is given, only the first **`maxsplit`** occurrences are replaced.

For example, **`$replace('abcdefabcde', 'abc', 'xyz', 1)`** returns "xyzdefabcde".

$substr()

`$substr(s, i [, j])` retrieves the substring in string s from position i to j. If i is negative then the substring begins that number of characters from the end of the string. If **`j`** is negative then the substring is from **`i`** up to **`j`** characters from the end of

the string. If the optional **j** is omitted, the substring ends at the end of the string. Some examples:

```
<esi:assign name="a_str" value="'whether tis nobler in the mind'"/>

 $substr( $(a_str), 0, 7 )   == "whether"
 $substr( $(a_str), 12, 6 )  == "nobler"
 $substr( $(a_str), 22 )     == "the mind"
 $substr( $(a_str), 0, -5 )  == "whether tis nobler in the"
 $substr( $(a_str), -8, 3 )  == "the"
 $substr( $(a_str), -8, -4 ) == "the"
```

$lower()

$lower(s) Returns an all lowercase version of the string **s**.

For example, **$lower('MakeLower')** returns "makelower".

$upper()

$upper(s) Returns an all uppercase version of the string **s**.

For example, **$upper('MakeUpper')** returns "MAKEUPPER".

Other Functions

$dollar()

This takes no parameters and returns a literal "$."

Example using **esi:assign** to set a user-defined variable (page 51):

```
<esi:assign name="dollar_sign" value="$dollar()"/>
<esi:assign name="a_hundred_dollars">$dollar()+'100.00'</esi:assign>
```

When the variable is evaluated, **$(a_hundred_dollars)**, it returns "$100.00."

$dquote() | $squote()

The function, **$dquote()**, returns a literal " (double quote), and **$squote()** returns a literal ' (single quote).

$int()

$int() explictly casts a string to an integer representation of the string, or to zero (0) in the case of an error. For example, the following value evaluates to the integer, 14.:

```
<esi:assign name="total" value="7 + $int('7')"/>
<esi:vars>$(total)</esi:vars>
```

You can use this to convert values passed as strings, such as query strings, to integers, where appropriate. For example, in the following example, the query string value "d" is "9," and using $int(), the nextday variable evaluates to 10.

```
<esi:assign name="day" value="$(QUERY_STRING{d})" />
<esi:assign name="nextday" value="$int($(day)) + 1" />
<esi:vars>$(nextday)</esi:vars>
```

Without **$int()**, the variable would evaluate to "91." The "1" would be implicitly coerced to a string, as described on page 66, and then concatenated to the "9."

$str()

$str() takes any input and will attempt to return the string representation of that input. This is most often used to cast integers to strings in cases where expressions would evaluate through addition when concatenation is wanted. For example, in this example, **UserID** is an integer:

```
<esi:assign name="cookie" value="$str($(UserID)) + $str($rand())'"/>
<esi:vars>
$add_header( 'Set-Cookie', $(cookie) )
</esi:vars>
```

Without **$str()**, the number, **UserID**, would be added to the random number **$rand**. With **$str(),** the result can be a cookie with the string form of **$rand()** appended to the string form of **$(UserID).**

It is not necessary to cast both numbers to a string because of the implicit conversion rules, but it is good to be explicit.

$len()

This accepts the name of a list, string, or dictionary (see page 51) and returns the length of the sequence: number of list items, length of a string, or number of name-value pairs in a dictionary. When used on an integer, the return value is undefined.

For example, **$len("longword")** would return 8, and if "listname" is the name of a list containing 12 items, then **$len($(listname))** would return 12. Another example of $len() is found in the example for the next function, **$list_delitem()**.

$bin_int()

$bin_int() takes a signed 4 byte integer and converts it into a 4 byte binary form. It does this by reading 8 bit segments of the int from least significant byte to most significant byte and outputting the binary representation of each byte. **$bin_int(x)** is equivalent to Perl's **pack("Na4", x)**;.

$list_delitem()

$list_delitem(sequence,index_#) accepts a list and an index number (an integer representing the position of the list item. The first position in the sequence is 0. This function deletes the value stored at the integer position, and it returns the value stored that was deleted. The list is now modified—the item has been deleted. For example:

```
<esi:assign name="my_car_list">
['Ferarri', 'Aston Martin', 'Toyota', 'Porsche']
</esi:assign>
<esi:assign name="not_belong" value="$list_delitem($(my_car_list),2)"/>
```

The first **assign** statement creates a variable, "my_car_list," that has four members. The second **assign** statement creates another variable, "not_belong," based on the **$list_delitem()** operating on "my_car_list" to return "Toyota," the item in position 2 which was deleted from the list.

Used iteratively, this function provides for looping.

A Loop Example Using a Query

Here's an example of the iterative use of **`$list_delitem()`**. Note that this example makes use of a variable, a member of a query string, as a list value. If a query string has the same variable defined more than once, ESI automatically creates a list.

Note** that this function is not intended to be a full iterative function. The ESI iterative statement,* **`esi:foreach`, is described on page 34.*

This example shows a member of the query string **`$(QUERY_STRING{'x'})`** that represents four items with the same name but different values, for example, **`?x=1&x=2&x=3&x=4`**. The four items could be four check boxes with the same name. The query value **`?x=1&x=2&x=3&x=4`** is the equivalent of a list created with **`esi:assign`** with the value **`"['1','2','3','4']"`**. Since query strings are always interpreted as strings, the results are strings.

In this example, then, the **`esi:assign`** statement (page 51) is used to create a variable, **`check_list`**, that accepts as its value the results of the query string.

Another variable, **`iter`**, is assigned as the value of **`$list_delitem`** operating on **`check_list`** to delete the first item and return the remainder of the list.

Finally, the **`choose`** block is used to test that the length of the list is greater than 0. When the test is met, the **`include`** statement fetches itself, in the sense that it calls the file it's in. This loop continues until the last item is deleted from the **`check_list`** and the length of **`iter`** becomes 0.

Note that in the **`choose`** block, the **`$len()`** function is used to determine the remaining number of items in the list.

Finally, note that to iterate as shown here, you could create a list using the **`assign`** statement without the query string, but that might not be as useful in this context.

Example:

loop.html

```
<html>
<body>
This is a loop test
<esi:assign name="check_list" value="$(QUERY_STRING{'x'})"/>
<esi:include src="loop_body.html"/>
</body>
</html>
```

---

loop_body.html

```
<esi:assign name="iter" value="$list_delitem($(check_list), 0)"/>
<esi:vars>
This is iteration number #$(iter)<br>
</esi:vars>
<esi:choose>
  <esi:when test="$len($(check_list)) > 0">
  <esi:include src="loop_body.html"/>
  </esi:when>
</esi:choose>
```

Note the QUERY_STRING in the request
127.0.0.1:3128/esi/loop.html?x=1&x=2&x=3&x=4
and the result...

```
<html>
<body>
This is a loop test
This is iteration number #1<br>
This is iteration number #2<br>
This is iteration number #3<br>
This is iteration number #4<br>
</body>
</html>
```

$rand() | $last_rand()

$rand([*n*] **)** generates and embeds a random number between 0 and (n-1). For example, given **$rand(1000),** the random number generated is between 0 and 999. The integer *n* is optional, and if it is omitted the number generated is between 0 and 99999999. For example:

```
<esi:vars>
<img src="http://myplace.net/ad/news.com/sz=392x72;ord=$rand()?"/>
</esi:vars>
```

http://myplace.net/ad/news.com/sz=392x72;ord=$last_rand() returns the last number generated by **$rand()**. For example:

```
<esi:comment text="use the same ad as before"/>
<esi:vars>
<img src="http://myplace.net/ad/news.com/sz=392x72;ord=$last_rand()?"/>
</esi:vars>
```

The generated number is passed from template pages to fragments, but not from fragments to templates. That is, if you use **$rand()** to generate a number in a parent page, you can use **$last_rand()** to recall the number in the child.

An Alternative to $last_rand()

Note that you can easily assign a **$rand()** value to a variable. For example:

```
<esi:assign name="LRand" value="$rand(10000)"/>
```

You could then use **$(LRand)** to return the number generated by **$rand()**, with an obvious advantage being that you can create multiple variables storing different random numbers.

$is_empty() | $exists()

These functions provide the ability to determine whether a variable or user-defined variable contains no data or if it exists at all. **$exists()** returns a true (1) if the object exists; **$is_empty()** returns a true if the object exists but consists of no data.

```
<esi:choose>
  <esi:when test="$exists($(HTTP_COOKIE{'group'}))">
      <esi:include src="http://www.akamai.com/a.html"/>
  </esi:when>
  <esi:when test="$is_empty($(HTTP_COOKIE{'group'}))">
      <esi:include src="http://www.akamai.com/b.html"/>
  </esi:when>
  <esi:otherwise>
      <esi:include src="http://www.akamai.com/c.html"/>
  </esi:otherwise>
</esi:choose>
```

$add_header()

$add_header(header_name,header_value) adds an HTTP response header; it does not replace existing headers. This can be virtually any header, but note that EdgeSuite does not pay attention to these headers as it passes them to the client. A header set on a fragment is included with response headers on the template. You can use this function to affect the downstream environment but not the EdgeSuite environment. An example:

```
<esi:vars>
$add_header('Set-Cookie', 'MyCookie1=SomeValue;')
$add_header('Set-Cookie',
      $url_encode('MyCookie2')+'='$url_encode('SomeValue2')+';')
<esi:comment text="the next statement should not contain line breaks,
      even though it may appear to have breaks in this example."/>
$add_header('Set-Cookie',
      $url_encode('MyCookie3')+'='$url_encode('SomeValue3')+';'+
      expires+'='+$http_time($time()+36000)+';' path=/;
      domain='+$(HTTP_HOST)+';')
</esi:vars>
```

Note the recommended usage of the **$url_encode()** function on both the cookie name and value, but not the equal sign between them.

Results:

```
Set-Cookie: MyCookie=SomeValue;
Set-Cookie: MyCookie=SomeValue3; expires=Fri, 20 Jul 2001 02:29:22 GMT;
      path=/; domain=127.0.0.1;
Set-Cookie: MyCookie=SomeValue2;
```

This example demonstrates the use of the **add_header()** function to set a Set-Cookie header. It also demonstrates what happens if you set multiple headers with the same

name—they do not replace headers of the same name, but instead they get added to the header list and are all sent. The order they are sent to the client is undefined and is not determined by the order they were set. HTTP places no precedence on headers arriving in a particular order.

$set_response_code()

This function allows you to send to set arbitrary response codes in the template or fragment. The form is **`$set_response_code(int [,html])`**, where *int* is a legitimate HTTP response code, and *html* is HTML code that may optionally be included with the response. An example: **`$set_response_code(404)`**.

If you use **`$set_response_code()`** to send an error code, note that EdgeSuite doesn't preserve the body of the message. You can use the optional *html* parameter to send back a custom error page. If you don't use the *html* parameter, a standard error page is generated, not the resulting ESI page. An example of code using the html parameter.

```
<esi:assign name="errorpage">
  <HTML>
  <BODY>
  This is the body of my 404 error page.
  </BODY>
  </HTML>
</esi:assign>

<esi:choose>
  <esi:when test="$(error)==1">
      $set_response_code( 404, $(errorpage) )
  </esi:when>
<esi:choose>
```

In this example, if the test for **`$(error)`** evaluates to true, the 404 response code is returned along with the **`$(errorpage)`** text which replaces the original output.

$set_redirect()

This function allows you to set a redirect (a 302, Moved Temporarily). The form is **`$set_redirect (Location)`**, where the string, *Location*, is the URL to redirect to. Because some browsers do not handle relative URLs correctly, the location should be absolute. Also, you may want to use **`$url_encode()`** function to remove the possibility of malformed URLs.

Note that if you use **`$set_redirect()`** to issue a redirect, you should *not* use the previous function, **`$set_response_code()`**, to set a redirect response code. The **`$set_redirect('http://xyz.example.com/')`** function is equivalent to **`set_response_code(302)`** plus the additional function, **`add_header('Location', 'http://xyz.example.com/')`**.

- With **`$set_redirec()`**, ESI removes the body of the response, creates a Location: header and passes this Location header to the template (from the fragment if it's created there). The implication is that if there is a redirect set on *any* fragment or the template in an ESI construction, the client browser will follow the redirect and the end user won't see the original ESI-produced page. In other words, you can't use **`$set_redirect()`** to replace a fragment in the original page.

The following example checks for the existence of a specific cookie. If the cookie exists, include the regular html fragment. If the cookie doesn't exist, redirect the client browser to another address in the same domain.

```
<esi:choose>
  <esi:when test="$exists($(HTTP_COOKIE{'group'}))">
      <esi:include src="http://www.akamai.com/regular.html"/>
  </esi:when>
  <esi:otherwise>
      $set_redirect('welcome.html')
  </esi:otherwise>
</esi:choose>
```

$add_cachebusting_header()

You can use this function without arguments to send a response header toward the client browser to prevent caching on downstream servers or proxies. See the discussion on page 22, or in the guide, *Time-to-Live in Cache: Methods and Considerations.*

$url_encode() | $url_decode()

In the standard URL, specified in RFC 1738, certain characters are not legal. **`$url_encode()`** accepts and encodes a string that may contain illegal characters. RFC 1738 can be found at:

`http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc1738.txt`

`$url_decode()` decodes special characters that were encoded from transmitting a URL. Mainly, this is used to decode the value of members of a query string. Note that EdgeSuite automatically stores the decoded version of query string values in the QUERY_STRING variable, when referenced by name **`($(QUERY_STRING{'foo'}))`**.

EdgeSuite 4.6 and Higher

The following character ↔ hex conversions are used for ESI code in EdgeSuite 4.6 and higher. (See below for legacy support).

CHAR	HEX	CHAR	HEX	CHAR	HEX
<	0x3C	|	0x7C	:	0x3A
>	0x3E	~	0x7E	;	0x3B
"	0x22	[	0x5B	?	0x3F
+	0x2B	]	0x5D	/	0x2F
%	0x25	‘	0x60	@	0x40
{	0x7B	’	0x27	=	0x3D
}	0x7D	space	0x20	?	0x3F

Example:

```
<esi:assign name="unencoded" value=This is a test ! !"#$%&'()*+,-./
      :;<=>?@[\]^_`{|}~+!
<esi:vars>
$url_decode($url_encode($(unencoded)))
$url_encode($(unencoded))
$url_encode($(NONEXISTANT|$(unencoded)))
$url_encode($(NONEXISTANT|'This is a default string.'))
</esi:vars>
```

This example creates a variable named "unencoded" with a value containing many illegal characters. The first line below the **esi:vars** statement will show the original string, since the statement decodes the encoded version. The second line shows the encoded result, as does the third line, which encodes $(unencoded) by default after not finding a variable named NONEXISTANT. The fourth line encodes "This is a default string." The results are seen below.

Results of the above sample **$url_** lines:

```
This is a test ! !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~+!
!%20!%22%23%24%25%26%27%28%29*+,-
      .%2f%3a%3b%3c%3d%3e%3f%40%5b%5c%5d%5e_%60%7b%7c%7d%7e+!
%!%20!%22%23%24%25%26%27%28%29*+,-
      .%2f%3a%3b%3c%3d%3e%3f%40%5b%5c%5d%5e_%60%7b%7c%7d%7e+!
This%20is%20a%20default%20string.
```

Legacy Support for Prior Versions

The following character ↔ hex conversions are used by default for ESI code prior to EdgeSuite 4.6. This method encodes the #, $, &, \, ^ ( ), characters not included in the 4.6+ version, and it does not include the plus sign (+), which is. If you are using this legacy encoding and want to upgrade to the 4.6+ URL encoding, you can do so by changing your EdgeSuite configuration:

CHAR	HEX	CHAR	HEX	CHAR	HEX
<	0x3C	|	0x7C	:	0x3A
>	0x3E	\	0x5C	;	0x3B
"	0x22	^	0x5E	?	0x3F
#	0x23	~	0x7E	/	0x2F
$	0x24	[	0x5B	@	0x40
%	0x25	]	0x5D	=	0x3D
&	0x26	‘	0x60	?	0x3F
{	0x7B	’	0x27	(	0x28
}	0x7D	space	0x20	)	0x29

Example:

```
<esi:assign name="unencoded" value="'#%$^&This is a test
      ><,.?[]{;:"|+=-_)('"/>
<esi:vars>
$url_decode($url_encode($(unencoded)))
</esi:vars>
```

This example creates a variable named "unencoded" with a value containing many illegal characters. The first line below the **esi:vars** statement will show the original string, since the statement decodes the encoded version. The second line shows the encoded result. The results are seen below.

Results of the above sample **$url_** lines:

```
#%$^&This is a test ><,.?[]{;:"| =-_)(
%23%25%24%5e%26This%20is%20a%20test%20%3e%3c,.%3f%5b%5d%7d%7b%3b%3a%22%
      7c+%3d-_%29%28
```

$html_encode() | $html_decode()

The **$html_encode()** encodes strings to HTML code and **$html_decode()** decodes HTML strings or digit codes to flat text. These functions are similar to the URL encoding discussed above, but unlike the URL encoding, these work asymmetrically.

Encoding

The **$html_encode()** function encodes only four special characters used in HTML for control purposes, and it leave others characters intact. Common characters don't need to be encoded unless they conflict with HTML control characters. The four characters encoded by **$html_encode()** are as follows:

Character	Encoding
> (greater than)	&gt
< (less than	&lt
& (ampersand)	&amp
" (double quote)	&quot

Decoding

The **$html_decode()** function can accept any HTML code and translate it into its original characters. The decoding is done following the ISO 8859-1 (Latin 1) character set. The codes 127–159 are not used; if encoded representations of these characters are found, they are left as is and not decoded.

Usage

An example use of this function would be to place HTML code, include the HTML tags, into a variable, and then recall the variable without having the browser read and act on the tags. For example:

```
<esi:assign name="foo">
$html_encode("<h1>This is a heading</h1>")
</esi:assign>
```

When "foo" is later reported outside an ESI block (**<esi:vars name="foo"/>**), the <h1> tags are encoded so that the browser doesn't create a major heading. The decode could be used still later to decode the string when formatted text is desired.

$base64_encode() | $base64_decode()

These functions encode/decode text using Base64 encoding. Both take a string to encode / decode and return the respective result.

$base64_encode(*text_to_encode*) takes any string and return its base64 representation.

$base64_decode(*text_to_decode*) works only with a properly encoded string. If there is an error decoding the string, **$base64_decode** returns an empty string.

Base64 encoding is symetrical, so the following is true

```
<esi:assign name="str" value="foo"/>
<esi:choose>
  <esi:when test="$(str) == $base64_decode( $base64_encode( $(str) ) )"/>
     true
  </esi:when>
</esi:when>
```

$digest_md5() | $digest_md5_hex()

These functions take a string and return a representation of the MD5 digest for that string. MD5 digests are asymetrical, meaning you cannot "decode" the MD5 as you can with Base64. An MD5 is a 128 bit integer, which can be accessed in ESI as either a list of 4 (32 bit) signed integers, or as a 32 character string representation. **$digest_md5(*text_to_digest*)** returns the a list of 4 (32 bit) signed integers, and **$digest_md5_hex(*text_to_digest*)** returns a 32 character string representation.

Time functions

There are three time functions: **$time()**, **$http_time()**, and **$strftime()**.

$time()

This takes no parameters and returns the current Greenwich Mean Time measured in the number of seconds since the Unix Epoch. For example, if the current time is Mon, 16 Jul 2001 14:36:56 GMT **$time()** returns "995319416."

$http_time()

$http_time() takes a single parameter, a integer representing Unix epoch time—that is, a number of seconds since Jan 1, 1970. This function returns a string formatted according to RFC 1123, the official method for representing time in HTTP. For example, **$http_time(995319416)** returns Mon, 16 Jul 2001 14:36:56 GMT.

OR, **$http_time($time())** would return the current time in the same format. A typical use of this construction is contained in the preceding example on the **add_header()** function.

RFC 1123 can be found at: **ftp://ftp.isi.edu/in-notes/rfc1123.txt**.

$strftime()

$strftime(time,format) This function accepts an integer representing Unix Epoch time and a format string, and returns a formatted string for the given timestamp or the current system time if no timestamp is given.

Ordinary characters placed in the format string are copied to the result without conversion. Conversion specifiers are introduced by a "%" character, and are replaced in the result as follows:

%a The abbreviated weekday name according to the current locale.

%A The full weekday name according to the current locale.

%b The abbreviated month name according to the current locale.

%B The full month name according to the current locale.

%c The preferred date and time representation for the current locale.

%C The century number (year/100) as a 2-digit integer.

%d The day of the month as a decimal number (range 01 to 31).

%D Equivalent to %m/%d/%y. This format is broadly used in the U.S., but is not recommended in an international context, since it can be confused with the more common form, %d/%m/%y.

%e Like %d, the day of the month as a decimal number, but a leading zero is replaced by a space.

%E Modifier: use alternative format. This is described below.

%G The ISO 8601 year with century as a decimal number. The 4-digit year corresponding to the ISO week number (see %V). This has the same format and value as %y, except that if the ISO week number belongs to the previous or next year, that year is used instead.

%g Like %G, but without century, i.e., with a 2-digit year (00-99).

%h Equivalent to %b.

%H The hour as a decimal number using a 24-hour clock (range 00 to 23).

%I The hour as a decimal number using a 12-hour clock (range 01 to 12).

%j The day of the year as a decimal number (range 001 to 366).

%k The hour (24-hour clock) as a decimal number (range 0 to 23); single digits are preceded by a blank. (See also %H.)

%l The hour (12-hour clock) as a decimal number (range 1 to 12); single digits are preceded by a blank. (See also %I.)

%m The month as a decimal number (range 01 to 12).

%M The minute as a decimal number (range 00 to 59).

%n A newline character.

%O Modifier: use alternative format.This is described below.

%p Either “AM” or “PM” according to the given time value, or the corresponding strings for the current locale. Noon is treated as “PM” and midnight as “AM.”

%P Like %p but in lower case: “am” or “pm” or a corresponding string for the current locale. (GNU[1])

%r The time in a.m. or p.m. notation. In the POSIX[2] locale this is equivalent to `%I:%M:%S %p'.

%R The time in 24-hour notation (%H:%M). For a version including the seconds, see %T below.

%s The number of seconds since the Epoch, i.e., since 1970-01-01 00:00:00 UTC.

1. GNU is a UNIX-like operating system with user-modifiable source code.
2. POSIX (Portable Operating System Interface) is a set of standard operating system interfaces based on UNIX.

%S The second as a decimal number (range 00 to 61).

%t A tab character.

%T The time in 24-hour notation (%H:%M:%S).

%u The day of the week as a decimal, range 1 to 7, Monday being 1. See also %w.

%U The week number of the current year as a decimal, range 00 to 53, starting with the first Sunday as the first day of the week 01. See also %V and %W.

%V The ISO 8601:1988 week number of the current year as a decimal, range 01 to 53, where week 1 is the first week that has at least 4 days in the current year, and with Monday as the first day of the week. See also %U and %W.

%w The day of the week as a decimal, range 0 to 6, Sunday being 0. See also %u.

%W The week number of the current year as a decimal number, range 00 to 53, starting with the first Monday as the first day of week 01.

%x The preferred date representation for the current locale without the time.

%X The preferred time representation for the current locale without the date.

%y The year as a decimal number without a century (range 00 to 99).

%Y The year as a decimal number including the century.

%Z The time zone or name or abbreviation.

%+ The date and time in date(1) format.

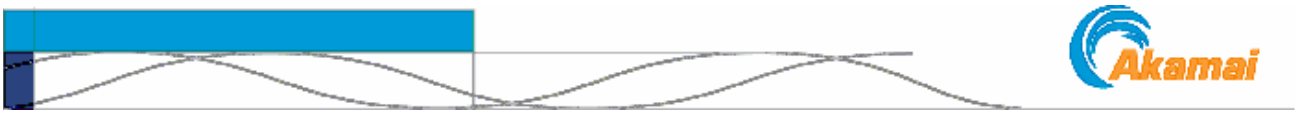
%% A literal "%" character.

Some conversion specifiers can be modified by preceding them by the E or O modifier to indicate that an alternative format should be used. The effect of the O modifier is to use alternative numeric symbols (e.g., roman numerals), and that of the E modifier is to use a locale-dependent alternative representation. If the alternative format or specification does not exist for the current locale, the behavior will be as if the unmodified conversion specification were used.

Example:

```
<esi:assign name="date_string" value="$strftime($time(), '%a, %d %B %Y
      %H:%M:%S %Z')"/>
```

This result of a **`$(date_string)`** evaluation at a particular moment, could be, e.g., "Mon, 16 July 2001 14:58:56 GMT".



CHAPTER 9. User-Defined Functions (Beta)

In addition to the functions described in the previous chapter, ESI provides, currently as a Beta feature, the ability to define custom functions[1].

Creating User-Defined Functions

A user-defined function is a named block of ESI code. It can be called to execute and to return a value for continued processing.

Properties

User-defined functions have the following properties:

- The function block can include any ESI code with the exception of **`esi:include`**, **`esi:eval`** and **`esi:function`**. You *cannot ne*st a function definition inside another function definition, and you cannot include a fragment.
- Normal ESI child-parent relationships are maintained with regard to namespaces. The parent's (caller's) namespace is visible to the functions in the child, but not *vice versa*.
- Recursion is allowed. The maximum call stack depth defaults to five; the number is configurable in your EdgeSuite configuration file.
- As is the case with the ESI provided functions, a user-defined function can be a part of another ESI expression.
- Parameters can be passed to the function as arguments to the function. The parameters are accessed with a single list variable inside the function.
- Functions are added to the namespace when the page is parsed, which means that functions can be defined anywhere on a single ESI page and can be referenced from anywhere on that page. This also means, however, that **`esi:eval`** statements that include functions must occur before the function is referenced since parsing of the eval statement does not occur until the tag is encountered during processing.
- Errors in function definitions cause a syntax error, resulting in the return of an HTTP 500 code.
- *Functions have no effect on the output of the ESI page.* Any content the function produces must be in its return value.

1. User-defined functions are extensions to he ESI 1.0 specification. See page 8.

The esi:function Block and its Usage

esi:function

The **function** statement block defines a custom function:

```
<esi:function name="fname">
  the value—the ESI code that comprises the function
  <esi:return value="exp"/>
</esi:function>
```

The **name** is composed of up to 64 alphanumeric characters (A-z, 0-9), and can include underscores (_) but cannot include a $ (dollar sign), which is reserved. The first character must be an alpha character of either case.

The block can include any ESI code with the exception of **esi:include**, **esi:eval** or **esi:function**.

The **return** statement is used to return a value from the function that may be used in an ESI expression.

The function can then be used anywhere ESI expressions such as variables and functions can be used. For example, after defining **fname** as shown above, the following will return its value:

```
<esi:vars>
  $fname()
</esi:vars>
```

or:

```
<esi:vars>
  <esi:assign name="xyz" value="$fname() + 7"/>
</esi:vars>
```

esi:return

A local function terminates when the end of the function body is reached or when an **esi:return** statement is processed. In the absence of an explicit return, a function return value is an empty string ("").

```
<esi:return value="exp"/>
```

Here, **exp** is the expression that is evaluated and returned to the caller. Two simple examples:

```
<esi:function name="return_one">
    <esi:return value="1"/>
</esi:function>

<esi:function name="return_one_in_a_silly_way">
    <esi:assign name="x" value="5"/>
    <esi:return value="$(x) - 4"/>
</esi:function>
```

A **return** statement can only appear inside a function body. If it appears anywhere else, it will cause an HTTP 500 code to be returned.

Arguments

Arguments to the function are available as a list variable, **`$(ARGS)`**. This variable contains all parameters referenced in the function calls argument list in a zero based order accessible by normal ESI list syntax. For example:

```
<esi:vars>$add( 5, 7 )</esi:vars>
<esi:function name="add">
     <esi:return value="$(ARGS{0}) + $(ARGS{1})"/>
</esi:function>
```

Or, in the following example, an **`is_odd()`** function is defined by evaluating the modulo for a number when divided by 2.

```
<esi:assign name="is_odd" value="$is_odd($(x))"/>
<esi:function name="is_odd">
    <esi:choose>
        <esi:when test="$ARGS{0} % 2 == 1">
            <esi:return value="1"/>
        </esi:when>
    </esi:choose>
    <esi:return value="0"/>
</esi:function>
```

Iteration

Since the **`$(ARGS)`** variable is a list, you can perform iterative operations. The following code returns a sum total for a list of numbers:

```
<esi:vars>$add( 5, 7, 29, 1, 5 )</esi:vars>
<esi:function name="add">
    <esi:assign name="sum" value="0"/>
    <esi:foreach collection=$(ARGS)>
        <esi:assign name="sum" value="$(sum) + $(item)"/>
    </esi:foreach>
    <esi:return value="$(sum)/>
</esi:function>
```

Recursion

The following example illustrates recursion calculation of an average for a list of numbers. This example defines two new functions, **addv** and **average**, in addition to the **add** function described in the prior example on iteration.

The **addv** function is needed to expand the list—that is, it takes all the items of the **ARGS** list and treats them as the first element—**$ARGS{0}**:

```
<esi:vars>$add( 5, 7, 29, 1, 5 )</esi:vars>
<esi:function name="addv">
    <esi:assign name="sum" value="0"/>
    <esi:foreach collection=$(ARGS{0})>
        <esi:assign name="sum" value="$(sum) + $(item)"/>
    </esi:foreach>
    <esi:return value="$(sum)/>
</esi:function>

<esi:function name="average">
    <esi:assign name="avg" value="0"/>
    <esi:assign name="len" value="$len( $(ARGS) )"/>
    <esi:choose>
        <esi:when test="$(len) > 0">
            <esi:assign name="sum" value="$addv( $(ARGS) )"/>
            <esi:assign name="avg" value="$(sum) / $(len)"/>
        </esi:when>
    </esi:choose>
    <esi:return value="$(avg)/>
</esi:function>
```

Recursion Inside an esi:when to Perform a "while" Operation.

This example function, **factor**, can factor an integer of indeterminate length. The function refers to itself to multiply an integer *n* by *(n-1)* until *n* reduces to 1:

```
<esi:function name="factor">
    <esi:choose>
        <esi:when test="$(ARGS{0}) == 1">
            <esi:return value="1"/>
        </esi:when>
        <esi:otherwise>
            <esi:return value="$(ARGS{0} * $factor($ARGS{0} - 1)"/>
        </esi:otherwise>
    </esi:choose>
</esi:function>

<vars>$factor(5)</vars>
```

The last line yields the result of factoring 5, that is,120.

Note that a recursion such as this is subject to the recursion limits discussed in under "Properties" on page 83.

CHAPTER 10. Internationalization

With some restrictions and caveats, any multibyte character set can be used in ESI[1]. The following sets are explicitly supported:

- Shift_JIS
- EUC-JP
- ISO-2022-JP

Detection

To use multibyte encoding, this feature must be enabled in your EdgeSuite configuration. You can specify the type of encoding (a) in your EdgeSuite configuration file or (b) with an HTTP Content-type header, that is,

```
Content-type: text/html; charset=charset_name
```

If the charset is unknown or there is no charset indicated, it is assumed to be a single-byte charset. If there is a conflict between the EdgeSuite configuration setting and the HTTP header setting, the HTTP header setting is used.

Restrictions

The following restrictions apply when using multibyte sets in ESI:

FUNCTION OR ELEMENT	LIMITATION
CGI Variables (HTTP Headers)	See discussion below, "Handling CGI Variables."
User-defined variable names (**esi:assign** and **esi:set**)	Variable names can only contain the ASCII characters [A-Z a-z 0-9]
The **$list_delitem()** function	Does not work on multibyte characters
Printing a dictionary or list. E.g., **<esi:vars>$(listname)</esi:vars>**	Does not give desired results when used with multibyte or high ASCII characters.
Booleans: **$has_i()** **$matches()** **$matches_i()**	Using a multibyte character set, these operate correctly only on US-ASCII characters. Using a single-byte character set, these work on all single-byte characters.

1. Internationalization extends the ESI 1.0 specification. See "The ESI Specification" on page 8.

Handling CGI Variables

When using multibyte encodings, you need to take care when you access data in the HTTP header or POST data. When you pass characters encoded in a character set other than US-ASCII to and from your web application, you must explicitly tell ESI which data needs to be transcoded from the source encoding to ESI's internal encoding and *vice versa*.

Common situations meriting attention are accessing cookie data or accessing data passed in the query string or from a POST body. For example:

```
<esi:vars>
Hello, $(QUERY_STRING{'name'}) <br>
</esi:vars>
```

The query variable, **`$(QUERY_STRING{'name'})`**, could be Shift_JIS or it could be ASCII; ESI has no way of knowing, and it will always use its own internal encoding.

Additionally, data in an HTTP header has probably been URL-encoded, a fact that needs to be taken into account when you code to access data.

ESI provides URL-encoding and decoding functions, described on page 76. For the basic transcoding needed between ESI's internal encoding and multi-byte encodings, ESI provides two language functions, **`$convert_to_unicode()`** and **`$convert_from_unicode()`**.

$convert_to_unicode() and $convert_from_unicode

These two functions are used only with multibyte encoding. When used in a single-byte environment, they will cause HTTP 500 errors.

`$convert_to_unicode(`*`text_to_convert`*`)` converts any multibyte text to ESI's internal encoding. For example:

```
<esi:vars>
Hello, $convert_to_unicode($(QUERY_STRING{'name'})) <br>
</esi:vars>
```

This tells ESI to convert the variable from the source encoding to its internal Unicode representation.

`$convert_from_unicode(`*`text_to_convert`*`)` converts ESI's internal encoding to the multibyte text listed in the EdgeSuite configuration or the Content-type header.

For example, in the following formulation, the coder may not intend to have the variable **`$(name)`** be a single-byte encoding.

```
<esi:assign name="name" value="ENCODED_TEXT"/>
<esi:include src="http://www.foo.com/tmp.pl?name=$(name)"/>
```

You can resolve this using the conversion function:

```
<esi:assign name="name" value="ENCODED_TEXT"/>
<esi:include src="http://www.foo.com/
      tmp.pl?name=$convert_from_unicode($(name))"/>
```

However, since byte values above 0x7F cannot appear in HTTP headers, you need to use the ESI function, **`$url_encode()`** (see page 78) to URL-encode the object:

```
<esi:assign name="name" value="ENCODED_TEXT"/>
<esi:assign name="urlname"
      value==$url_encode($convert_from_unicode($(name)))"
<esi:include   src="http://www.foo.com/tmp.pl?name=$(urlname)"/>
```

CHAPTER 11. Configuration & Content Control

This chapter is an overview to ESI configuration and content control considerations. This is not intended to cover the whole of the topic, but is meant to familiarize you with some of the methods and capabilities that apply specifically to ESI. For the broader and more detailed discussions, please see the following documents:

- The *EdgeSuite Configuration Guide* details the configuration and control options and parameters used for sites and objects in EdgeSuite, including ESI.
- *EdgeSuite Handling of Edge-Control & Other HTTP Headers* discusses the use of HTTP request and response headers in the EdgeSuite environment.
- *Time-to-Live in Cache: Methods and Considerations* discusses the various methods for determining the caching properties of objects on Akamai EdgeSuite servers.
- For information specifically on cookies and Session IDs and the use of these in ESI, see *EdgeSuite Session ID Support*.

Configuration and Control Mechanisms

There are four primary content control mechanisms for objects requested and served through EdgeSuite, three of which are discussed here: requests, configuration files, and response headers—specifically, the Edge-control header. [1]

- **Requests** can include a number of content control instructions. In ESI, instructions can be specified with attributes of the **`include`** statement such as **no-store**, **dca**, **ttl**, and others, discussed in Chapter 3 starting on page 15.
- **EdgeSuite configuration files** provide the broadest and deepest control. These are the primary method for setting options across groups of objects—on a per-directory basis, on file extensions such as .html, .gif, .jpeg, etc., or based on matching other criteria. For example, you can set up a match so that if a query contains a particular string, the request should be handled in one way; otherwise, do something else.

  EdgeSuite configuration files provide for setting broad default values as well as fine-tuning for particular cases. For example, you can nominate files for ESI processing, or you can specify TTLs (*time-to-live in cache*) for fragments that are different from the time specified for templates or other objects.

1. The fourth mechanism for specifying metadata for EdgeSuite objects, the v1 ARL (Akamai Resource Locator), is not discussed here.

EdgeSuite configuration files are distributed to every relevant Akamai server and are not meant to be updated frequently. These files are created and managed by Akamai based on your requirements and specifications.

- The **Edge-control header** (formerly known as the Ak-control header) is a form of HTTP response header, included in addition to the normal HTTP response header. An Edge-control header applies only to the associated object.

  For complete information on the Edge-control header and on HTTP header handling in EdgeSuite, see *EdgeSuite Handling of Edge-Control & Other HTTP Headers*.

  The following Edge-control header elements are of particular relevance to ESI.

  - **`dca`** nominates the request for Dynamic Content Assembly and it sets the processing type. This it is a required setting for every file to be ESI-processed by EdgeSuite; the configuration can be with the header or in the configuration. The setting for ESI is **`esi`**. The form in the header is: **`Edge-control:dca=esi`**). Setting this specific header tells EdgeSuite to parse the page for ESI language elements and process the relevant code. The other types allowed are **`xslt, java, akamaizer,`** or **`noop`**. The **`Edge-control:dca=noop`** form tells EdgeSuite *not* to process the content for ESI or for XSLT. A setting of **`noop`** tells EdgeSuite not to parse the content for ESI, for example, even if the file is used as a fragment and contains ESI code.
    You can daisy-chain processors by using the construction, **`Edge-control:dca=xslt->esi.`**
    This means first perform an XSL Transformation on the object, then process the output as ESI. You can daisy-chain the Akamaizer, XSLT and ESI processors; you can daisy chain up to five processors total, in any order.
  - **`dca-enable-debugging-tag`** enables the use of the **`<esi:debug/>`** statement to turn on debugging for the current page, its parents and any fragments. If this is not enabled (Off), the **`<esi:debug/>`** statement is ignored.
  - **`cache-maxage`** specifies a TTL (Time To Live) for the object in Akamai server cache. An **`cache-maxage=0s`** means that the object will be cached, but every time it is requested EdgeSuite will attempt to revalidate it.
  - **`no-store`** says the object cannot be cached, and existing cached instances are purged. Akamai retrieves the object from the origin upon every request.
  - **`bypass-cache`** is similar to **no-store**, but it passes the request without removing the underlying object from the cache if it's already in cache
  - **`must-revalidate`** instructs EdgeSuite not to serve a stale cached object if it cannot revalidate that object. By default, Akamai servers will serve a potentially stale cached object if they are unable to revalidate that object because of, for example, loss of connectivity to the origin server. The **`must-revalidate`** header overrides that default.

  You can also set TTL settings using the HTTP Cache-Control or Expires headers, *if* the use of these headers is enabled in your EdgeSuite configuration file.

Order of Precedence

If EdgeSuite receives contradictory instructions from different content control mechanisms, it uses the following order of precedence: (1) requests, (2) responses, and (3) configuration files. That is, settings in requests override those in the response header, which in turn override settings in the configuration files.

For example, if an individual file's Edge-control **`cache-maxage`**, is set to 30 minutes, while the configuration file sets a 15-minute maximum on the same request, the Edge-control header 30-minute setting prevails. But an **`include`** request for the object containing a 1-hour **ttl** attribute prevails over the Edge-control header setting.

The Matching Criteria

In EdgeSuite, you can use a variety of criteria to determine which requests shall be affected by the content control attributes you want to apply—that is criteria that are used to determine the scope of the application of attributes.

As described in the *EdgeSuite Configuration Guide*, the criteria can be as simple as the file name or directory, or can as complex as matching a portion of a string extracted from a cookie, header, or query string. The scope can be the entire web site or a single object when certain conditions apply. Every configuration uses one or more of these matching criteria, and certain criteria, such as a default **maxage** setting, are required.

The ESI Fragment Match

Configuring ESI can involve the use of any number of different matching criteria. There is one that is geared exclusively to ESI, and that is: you can set other attributes based on whether the requested object is an ESI fragment. For example, you can set a specific **maxage** on requests for fragments only.

Options and Attributes

The following are options or attributes you set based on the object matching some criteria—for example, whether the object is an ESI Fragment or not, or whether the a response header contains specific data.

Enable ESI / XSLT Through Response Headers.

This enables the use of the Edge control response header, **`Edge-Control:dca="esi"|"xslt"|"akamaizer"|"java"|"noop"`** to nominate objects for ESI or XSLT processing (see page 90). This and the next option (**dca**) are the two acceptable ways of enabling DCA processing. Enabling response headers provides per object control.

Dynamic Content Assembly

This nominates an object for (a) ESI, for the (b) Edge Transformations Service, or (c) specifies no DCA processing. This and the previous option are the two acceptable ways of enabling DCA processing. While the previous option enables use of response

headers to nominate objects, this option nominates the objects that match some criteria for whatever processing is specified. When using ESI or XSLT, this should not be set to match an entire site, but should only apply to the specific text/html files that are to be processed. You can daisy-chain up to five ESI and/or XSLT processors.

Enable Internationalization Features

You can enable the use of multi-byte character sets, which also impacts certain other ESI features, as discussed in "Internationalization" on page 87.

Set Content Type

When you use the internationalization features in ESI, you need to specify the character set to use, either in EdgeSuite configuration or with a Content-Type header. See "Internationalization" on page 87.

Disable iteration.

You can disable the use of the iteration (**`<esi:foreach>`**) within your ESI code.

Enable the Debugging Statement.

When turned on, this enables the use of the **`<esi:debug/>`** statement to turn on debugging for the current page, its parents and any fragments. If this is not enabled (Off), the **`<esi:debug/>`** statement is ignored. This option has an Edge-control response header analog: **`Edge-control: dca-enable-debugging-tag`**.

Disable the backslash escape function

This disables the backslash (\) function as an escape character, described on page 59.

Disable function errors

If "Off," which is the default, errors on functions—functions that don't exist or which use improper parameters—will generate a 500 error, and no processing will take place. If "On," function errors are ignored.

Disable ESI processing of POST responses

By default, you can use POST responses from the origin as discussed on page 46. This function can be disabled.

Enabling and Disabling the Passing of EdgeScape and EdgeScape Pro data.

See the discussion about the GEO data on page 47. These options can be used only if you subscribe to EdgeScape or EdgeScape Pro. You can enable passing the data forward to your origin server *and* to ESI, or you can enable to passing the data forward to your origin server *but not* to ESI.

Disable Strict Processor Domain Checking when Processor is "None."

By default, EdgeSuite verifies that, if a processor type (**dca=**) of "none" is specified in an ESI **`include`**, the domain of the template page matches the domain of the included **`src`** object. This option disables this domain check.

Disable Strict Stylesheet Domain Checking

By default, EdgeSuite prohibits specifying an XSLT stylesheet in an ESI **`include`** from a different domain from the XML page. This option disables this check.

Inherit the Processor Type of the Parent

This instructs EdgeSuite to process DCA fragments using the same process as the template page. For example, if EdgeSuite is processing an ESI template, treat the included object as an ESI fragment. This does not apply to XSLT; when the template is XSLT, the child is always processed using XSLT.

The default behavior is for fragments with no processor type specified to receive no processing at all. With this option set to "On," fragments that have no processor type specified for them (either through the configuration file or response headers) inherit the processor type of the template page. If the processor type for the fragment is specified in the response headers or in the configuration file, this option has no affect.

Disable the Computing of Downstream TTL

This option disables the computation of downstream TTL.

By default, ESI computes the appropriate TTL (time-to-live in cache) for the response to the client by comparing the caching metadata associated with all the fragments. If a **ttl** attribute is present (and there are no instructions to prohibit caching), the value will apply to the object to which it is associated, and in addition, the root response header will be set to no more than the least **ttl** value of all objects associated with the template. In other words, the resulting ESI page will be cached downstream for a period defined by the shortest **ttl** of all the objects that compose the page.

Set the Downstream TTL

You can set the value of the HTTP Cache-control header on ESI-assembled objects to client. You can set a **no-store**, **no-cache**, or a positive integer specified with s (seconds), m (minutes), h (hours), or d (days) (e.g., 10h).

When you use this option, you should make sure that the honoring of HTTP Cache-control headers is also enabled in your EdgeSuite configuration file. This will avoid potential errors.

If you use **no-store**, the header is set to **Cache-control: no-store, no-cache** and **Expires: now**, in order to purge downstream caches.

When set on a fragment, this value is used as one of the values used by ESI to calculate the root TTL value determined by the computation of downstream TTL. When you set a downstream TTL value in this manner on the template page, this value overrides other TTL settings (but not a **no-store** or **bypass-cache**) calculated as described in the preceding subsection.

When you allow ESI to calculate the downstream TTL, the value it sends will account for time already spent in cache. But when you use the Downstream TTL attribute, neither Cache-control: max-age setting nor the Expires data can be updated to account for time already spent in cache. See the discussion on page 22.

Cache ESI Results

By default, the ESI resulting objects are not cached, but you can set up result caching through EdgeSuite configuration.

Enable ESI to Pass Cookies

This option causes ESI to pass any Set-Cookie headers (received from the origin server) to the fragment request generated by the ESI code. This is important for sites that perform use session ID's in cookies, as it prevents the fragment requests from going forward without the session cookie and thus generating a new user session on the origin server. ESI passes the value of Set-Cookie headers coming from the origin server into fragment requests as Cookie headers. For information specifically on cookies and Session IDs and the use of these in ESI, see *EdgeSuite Session ID Support*.

Pass a Bandwidth Usage Variable for Use in ESI

You can have defined a variable representing the bandwidth used in serving content to end users. The variable, which can be used in ESI (page 48), is defined as follows:

- a name
- a CP Code or range of CP Codes (the CP Code is a unique number or set of numbers associated with your Akamai account contract.)
- for the CP Code or CP Code range, (a) the current Megabit usage per second, or (b) the sum of MB usage since the beginning of the current calendar month.

Extracted Values

You can extract values from cookies, path components or parameters, or query strings, and pass them to ESI as variables. A key use of this is to use cookies in maintaining session IDs. This extraction is discussed in the *EdgeSuite Configuration Guide* and in the *EdgeSuite Session ID Support* guide.

Use this XSLT StyleSheet

This option can be used to specify the file to be used as the stylesheet in an XSLT transformation. This can be any path or URI, but the string cannot begin with a dot (.) or two dots (..). By default, EdgeSuite uses the path specified in your configuration file as the base path for the relative path it uses in identifying the stylesheet. You can change this and use the URL of the current directory of the XML file as the base directory.

A stylesheet set with an ESI **`include`** statement takes precedence over a stylesheet specified in the EdgeSuite configuration file, but this configuration setting takes precedence over a stylesheet set in the XML object's code.

Enable Bitwise Operators

You can opt to use bitwise operators, as described on pages 60 and 64.

Use the Pre-4.6 URL-Encoding

You can opt to use the pre-EdgeSuite 4.6 URL-encoding function. See "$url_encode() | $url_decode()" on page 76.

Use the Pre-4.8 esi:try Block Behavior

You can opt to use the pre-EdgeSuite 4.8 behavior on exception handling in an **esi:try** block. Prior to 4.8, only HTTP 200 responses were considered successful responses, and other responses triggered the except statement. As of 4.8, no exception is triggered if an HTTP 200, 301, 302, or 401 response is received. See "try | attempt | except" on page 42.

CHAPTER 12. Exception and Error Handling

Overview

When ESI fails to process and serve a page, there are a number of potential causes. These are the conditions you may want to investigate:

- ESI code errors: ESI syntax errors or typos, or incorrect path or file specifications for objects.
- Problems with the fragments you're trying to fetch: tags that duplicate or conflict with template tags or other fragments; HTML that isn't well-formed.
- Errors or conflicts in metadata settings: for example, your configuration file doesn't specify the objects for ESI processing. Or there is no **`Edge-control: dca=esi`** header associated with an object when one is required.
- Limits exceeded: Too many **`includes`** on a page, or the file size total is too large.

Using the Debugger

You can run and test your ESI pages using ESID, the ESI debugger. This is covered starting in the *EdgeSuite ESI/XSLT Development Tool*.

Error Messages

When in regular (not debug) mode, if EdgeSuite fails to serve the results of ESI processing, it serves one of two error messages to the end user:

- In the case of parsing or syntax errors, a HTTP 500 server error message.
- In the case of file errors—files not found—an HTTP 404 Not Found message.

Mechanisms for Handling

There are two control mechanisms for exception and error handling in EdgeSuite: EdgeSuite configuration and certain ESI language elements. You can make use of both mechanisms simultaneously, one at a time, or neither, depending on your needs. This section briefly describes these mechanisms and offers some examples.

Configuration Data and Default Objects

The first mechanism is to use default objects to replace not-found objects by setting **`failure-action`** attributes in the EdgeSuite configuration as described briefly in "Configuration & Content Control" on page 89. The default object can be whatever serves your needs: for example, a message to the user including alternative links, or a simple 1x1 pixel GIF. You can set defaults selectively to apply to particular sets of files: to a template, to a template and fragments, not to the template but to some fragments, etc.

EdgeSuite can process ESI markup in the default object.

The general EdgeSuite behavior is that if a default is specified, retrieved, and cached, and EdgeSuite cannot fetch the requested file because of an HTTP 500, 503, or 504 code, it serves the default object instead of an error. For other errors, it returns the error received from the origin server (unless there is some other action specified in the configuration or code).

However, with respect to ESI, if EdgeSuite has already fetched a template and the failure is on the fragment, the behavior depends on how you have structured the EdgeSuite configuration and the ESI code.

There are three general cases:

- When there is a default object specified for the template.
- When there is no default object specified for the template.
- Where there is no default for either the template or the fragment.

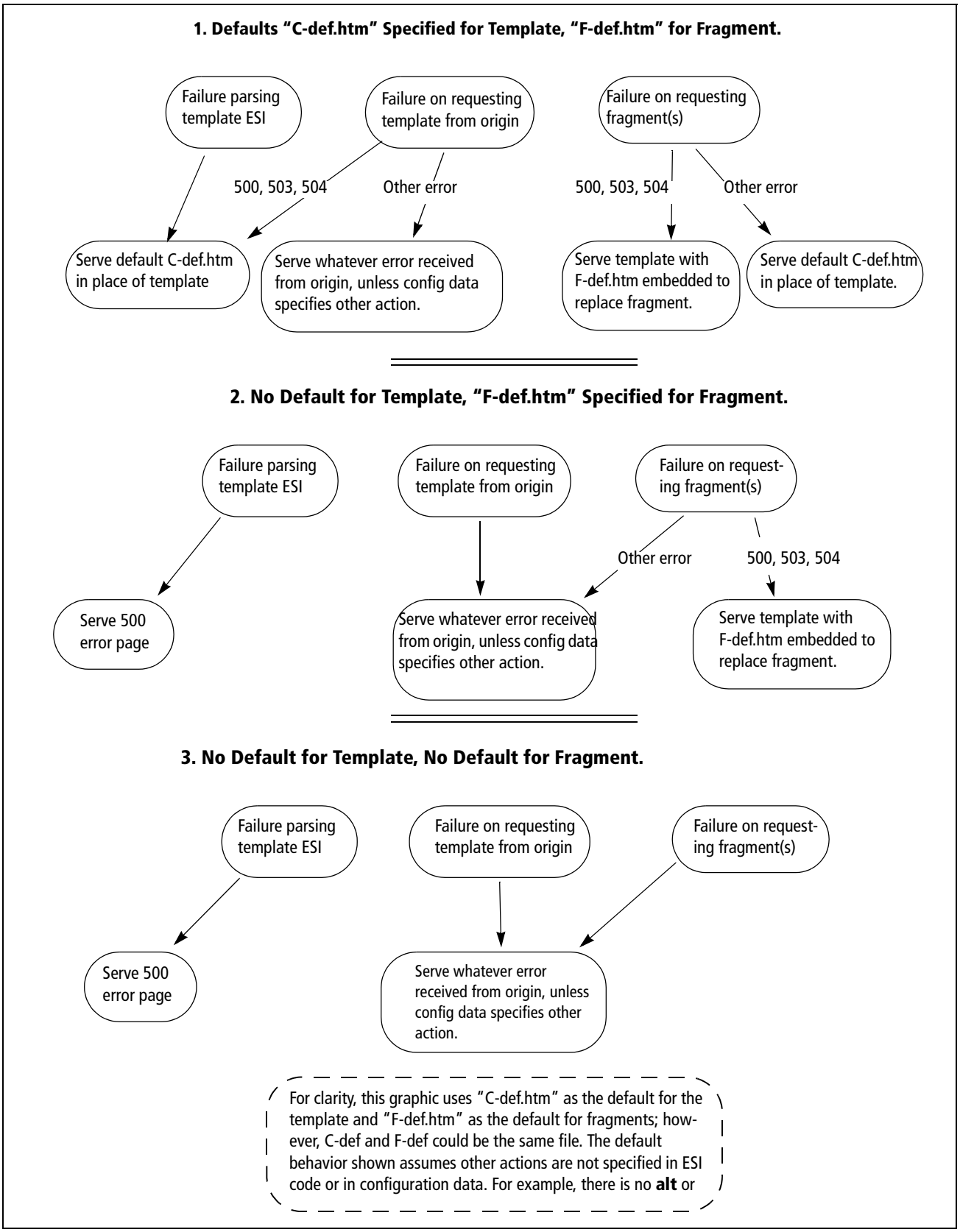


Figure 3. Using Default Objects with ESI

Example: Default Pages Set in Configuration

An Akamai server has a problem connecting to your site, the *origin site*, to refresh the contents of a simple template HTML page for which the TTL (time-to-live) has expired. You have instructed EdgeSuite not to serve stale content (see "**must-revalidate**" on page 90).

If there is no default object specified for the template, EdgeSuite in most cases serves the error reported from the origin to the end user[1], unless configuration data specifies some other action.

However, in this example EdgeSuite has received a 500, 503, or 504 error, a default object is specified as a **failure-action** attribute in the EdgeSuite configuration files and the default has been saved in the EdgeSuite server's cache. EdgeSuite retrieves the default and serves it to the end user despite the inability to connect to the origin site.

Default for Template

To modify the example: let's say the template page is current—though there is a default cached, EdgeSuite doesn't need to use it.

However, the template includes a fragment that does need fetching. The template page is parsed by EdgeSuite and upon attempting to fetch the **include** URL, an error results. What happens next depends on the type of error:

- On a 500, 503, or 504 error, EdgeSuite replaces the fragment with the default.
- On an error other than the above (a non-200 code response), EdgeSuite serves the default for the template in place of the template.

No Default for Template

To modify the example in a different manner: the template doesn't need fetching, but there is *no* default specified for the template.

In this situation, if there is an error attempting to fetch the fragment:

- On a 500, 503, or 504 error, EdgeSuite replaces the fragment with the default.
- On an error other than the above (a non-200 error response), EdgeSuite serves a 404 error message to the user.

If there is no default for the fragment, and if the ESI language controls do not specify an exception action, EdgeSuite generates a 404 error message.

1. One exception: you can instruct EdgeSuite to serve the stale content rather than no content at all.

ESI Language Control

The second mechanism is found in the ESI language, which furnishes two specific elements that provide fine grain control over page assembly in error scenarios:

- the **onerror** attribute of the **include** statement.
- the **try | attempt | except** block

These elements are described on pages 18 and 42, respectively.

Example: Using onerror

You have HTML content with several ESI **include** tags. In serving a particular request, EdgeSuite has trouble fetching one of the includes. If the **include** tag does not include the **onerror** attribute, the situation is the same as that in the first example. That is, EdgeSuite triggers an error leading to one of two results: either a 404 error is returned to the end user or, if a default object for the URL is cached, the default is served to replace the fragment.

In this example, though, you have a group of advertisements along the right margin of the page, and it's more important to serve the page without one ad than to serve an error page with no ads. You don't want a default or an error; you want simply to skip the missing ad and display the rest of the page. You use the **onerror** attribute to tell EdgeSuite to just ignore this **include** if it fails.

The page assembly is completed and the final page (missing the one **include**) is served to the user.

The syntax for the above scenarios might look like this:

```
<table>
<tr>
<td>
The text in the left column... news and reviews....
</td>
<td>
<table>
<tr><td><esi:include src="http://www.foo.com/get_ad_1.html"
      onerror="continue"/></td></tr>
<tr><td><esi:include src="http://www.foo.com/get_ad_2.html"
      onerror="continue"/></td></tr>
<tr><td><esi:include src="http://www.foo.com/get_ad_3.html"
      onerror="continue"/></td>/tr>
<tr><td><esi:include src="http://www.foo.com/get_ad_4.html"
      onerror="continue"/></td></tr>
</table>
</td>
</tr>
</table>
```

Example: try Block

You want to fetch and display a non-cacheable fragment, such as an ad fragment, for which there is no default object. If the fetch fails, you can still include a link and text using the **`try`** block. The code might look something like this:

```
<esi:try>
<esi:attempt>
      <esi:include src="http://www.foo.com/get_an_ad.html"/>
</esi:attempt>
<esi:except>
      <a href="http://www.foo.com/contest.html">Win a Free Car!!!</a>
</esi:except>
</esi:try>
```

If the ad call fails, the user still sees, "Win a Free Car!!!" and is offered the link to the ad site.

CHAPTER 13. An Extended ESI Example

This simple example attempts to bring together some of the main ESI elements to illustrate their use. At the end of this section is a fuller listing of the code discussed in the example.

You can also view many examples on-line at http://esi-examples.akamai.com.

Let's say you are developing My.Place.com, a Web service that provides local news, sports, and entertainment information. You target users in localized geographic areas who have defined themselves as having specific interests.

Readers define their own local areas by town, city, urban community, or other geographic criteria. Readers customize their pages by choosing what content they want to see and the order in which they want to see it.

When the user customizes the page, they choose from a limited set of options that results in one of about two dozen formats, or form pages, which are pre-constructed. The two essential pieces of information written to the cookie are the form and the location they want to use.

The other content they'll see at the top of the content section of the page is an occasional news or offer from My.Place.com itself.

An end user from Exeter, USA, who is interested in Exeter news, sports, music, and movies, might see something like the page in Figure 4. The callouts 1, 2, and 3 relate the portions of the page here to the page form as it appears on a subsequent page, and to the narrative discussion and code fragments.

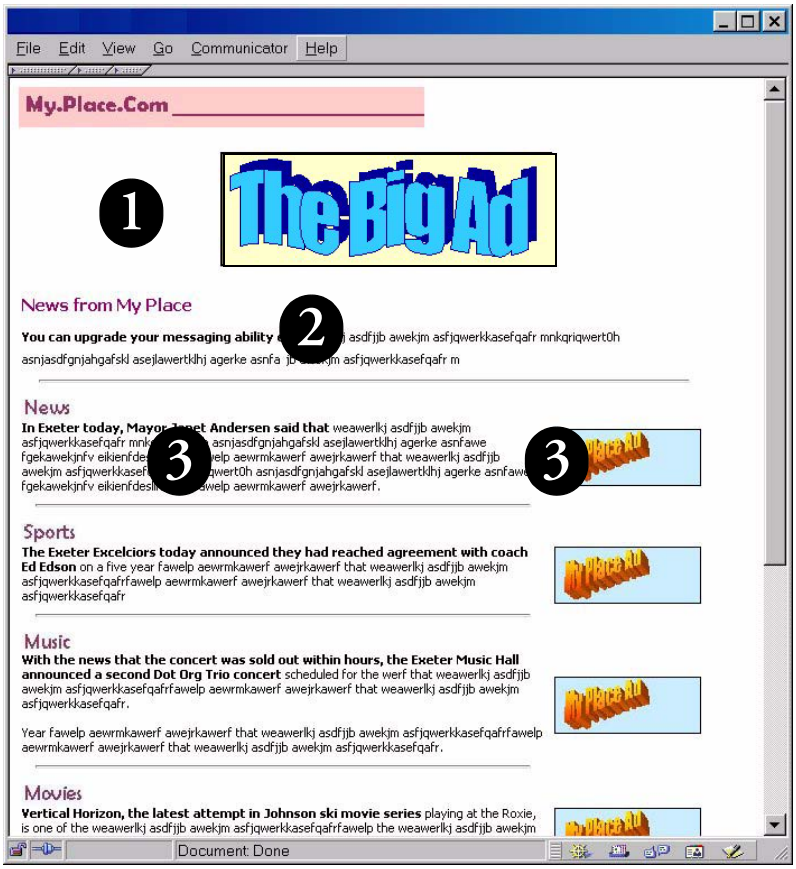


Figure 4. My.Place.com for a User from Exeter, USA

How to Build It

In building this page, the first thing you want to do is tell EdgeSuite which form and location to use, based on the cookie information.

You can do this with a **`choose`** block in a template page that isn't displayed to the user but, taking advantage of the ability of ESI to nest include statements, tells EdgeSuite to load the appropriate form template for the appropriate location.

The code might look like this:

```
<HTML>
<HEAD>
  <TITLE>MyPlace.com</TITLE>
</HEAD>
<BODY>
<esi:comment text="hide the ESI tags and select the template"/>
<!--esi
<esi:choose>
      <esi:when test="$(HTTP_COOKIE{'formtype'})=='type1'">
          <esi:include src="http://www.myplace.com/r4c5/templates/
              t1.html"/>
      </esi:when>
      <esi:when test="$(HTTP_COOKIE{'formtype'})=='type2'">
          <esi:include src="http://www.myplace.com/r4c5/templates/
              t2.html"/>
      </esi:when>
      <esi:when test="$(HTTP_COOKIE{'formtype'})=='type3'">
          <esi:include src="http://www.myplace.com/r4c5/templates/
              t3.html"/>
      </esi:when>
.
.
      <esi:otherwise>
          <esi:include src="http://www.myplace.com/templates/new.html"/
      >
      </esi:otherwise>
</esi:choose>
-->
</BODY>
</HTML>
```

Note that the **otherwise** in this construct says that if there is no form template chosen, this must be a new user or a user who is not logged in on their home machine, so offer up the subscription / login page.

This user takes a NewsSportsMusicForm1 (nsmf1) form as the main template page. Without the markup, it looks like Figure 5, with only the persistent objects showing. The "Ad Goes Here" and "Ad column" in the graphic do not appear in the form, but are added for information purposes only for the example.

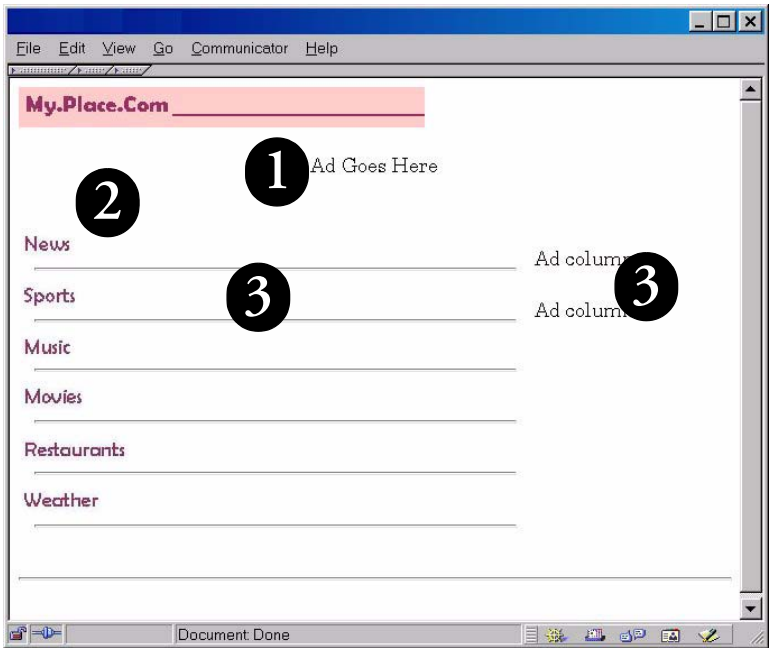


Figure 5. The nsmf1 form, the template page.

The news, sports, and so forth are in a two column table, with the right column holding niche ads and general ads—sports for sports, music for music, anything for news or weather, and so forth.

The Big Ad

1 The big ad at the top center rotates through different advertisers and is different each time the page is opened. But if for some reason the ad cannot be fetched, you have an ongoing backup ad for a free trip to Hawaii. For this you use the try block, and it may look something like this:

```
<P ALIGN="CENTER">
<esi:try>
      <esi:attempt>
          <esi:comment text="Include the big ad"/>
          <esi:include src="http://www.myplace.com/r4c5/ads/
      bigad.html"/>
      </esi:attempt>
      <esi:except>
          <esi:comment text="use alternate link if you don't find it"/>
              To Win a Free Trip to Hawaii,<a href="http://www.contest
                  limited.com/hawaii.html"> Click Here!!!</a>
      </esi:except>
</esi:try>
</P>
```

The My.Place News Row

2 The first piece of content is the "News from My Place." This is an occasional announcement or offer that spans both columns and has no ad in the right column. It is not always included, so you want to be able to skip it if it isn't there. In the row, you use a simple **include** with a **onerror**.

```
<TR>
<TD COLSPAN="2">
<esi:include src="http://www.myplace.com/r4c5/mpnews/top.html"
      onerror="continue"/>
</TD>
```

The Content Rows and the Smaller Ads

3 Unlike the "News from My Place," the News, Sports and other rows provide content you don't want to skip. If for some reason the current content is not available, you want to fetch the most recent previous content. If for some reason neither of these are available, you have used the EdgeSuite configuration files to set up a default message to the user that lets them know there is a problem and you appreciate their patience. You don't include a **onerror** in this statement because you don't want to simply skip the content.

In the right-hand column for each row is an ad. If the ad can't be fetched, you want to display an invitation to potential advertisers. So you use a **try** block for that.

Thus, the News row would look like this:

```
<TR>
<TD>
<!--esi
<esi:include src="http://www.myplace.com/r4c5/exeternews/
      20010128news1.html" alt="http://www.bak.myplace.com/r4c5/news/
      lastexeternews.html"/>
  -->
</TD>
<TD>
<esi:try>
      <esi:attempt>
      <esi:include src="http://www.myplace.com/r4c5/ads/news1.html"/>
      </esi:attempt>
      <esi:except>
      This spot is reserved for your company's advertising. For more
      information, <a href="http://www.biz.myplace.com/adpolicies/
      columnads.html"> click here!</a>
      </esi:except>
</esi:try>
</TD>
</TR>
```

The other rows have a similar construction—for example, the Sports row:

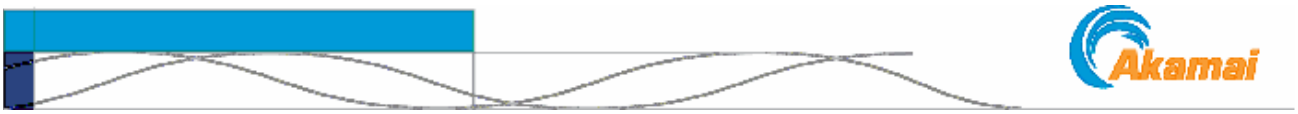
```
<TR><TD>
<!--esi
<esi:include src="http://www.myplace.com/r4c5/exetersports/
      20010128sports1.html" alt="http://www.bak.myplace.com/r4c5/
      sports/lastexetersports.html"/>
  -->
</TD><TD>
<esi:try>
      <esi:attempt>
      <esi:include src="http://www.myplace.com/r4c5/ads/sports1.html"/>
      </esi:attempt>
      <esi:except>
      This spot is reserved for your company's advertising. For more
      information, <a href="http://www.biz.myplace.com/adpolicies/
      columnads.html"> click here!</a>
      </esi:except>
</esi:try>
</TD></TR>
```

The Code Listing

It may be helpful to review a sample of the ESI-marked up HTML code. The following code is for the template shown in Figure 4 on page 102. This shows only the first content row, the "News," since the "Sports," "Music," and other content is redundant in terms of ESI construction.

```
<HTML>
<HEAD>
  <TITLE>My Place</TITLE>
</HEAD>
<BODY>

<esi:comment text="get the My Place image as a persistent object"/>
<P><IMG SRC="http://www.myplace.com/r4c5/templates/nsmf1/myplace.gif"
      WIDTH="362" HEIGHT="36" ALIGN="BOTTOM" BORDER="0"></P>

<esi:comment text="get the big ad or display the backup"/>
<P ALIGN="CENTER">
<esi:try>
      <esi:attempt>
          <esi:comment text="Include the big ad"/>
          <esi:include src="http://www.myplace.com/r4c5/ads/
      bigad.html"/>
      </esi:attempt>
      <esi:except>
          <esi:comment text="use alternate link if you don't find it"/>
              To Win a Free Trip to Hawaii,<a href="http://www.contest
                  limited.com/hawaii.html"> Click Here!!!</a>
      </esi:except>
</esi:try>
</P>
```

```
<esi:comment text="start the content table. First row gets the my place
      bulletin or skips it if not there"/>
<TABLE BORDER="0" CELLPADDING="2" CELLSPACING="1" WIDTH="94%">
<TR>
  <TD COLSPAN="2">
  <esi:include src="http://www.myplace.com/r4c5/mpnews/top.html"
      onerror="continue"/>
  </TD>
</TR><TR>


<esi:comment text="NOTE -- this begins block for content row">
<esi:comment text="Get the NEWS image, a persistent object"/>
  <TD WIDTH="77%"><IMG SRC="http://www.myplace.com/r4c5/templates/
      nsmf1/news.gif" WIDTH="46" HEIGHT="19" ALIGN="BOTTOM"
      BORDER="0"><BR>

<esi:comment text="Get the latest news content if present, or alt the
      last previous news. If both are missing, it's an error."/>
<!--esi
<esi:include src="http://www.myplace.com/r4c5/exeternews/
      20010128news1.html" alt="http://www.bak.myplace.com/r4c5/news/
      lastexeternews.html"/>
  -->
<HR ALIGN="CENTER" WIDTH="95%">
</TD></TR>
<esi:comment text="Get the ad for the news row."/>
<TR><TD>
<esi:try>
      <esi:attempt>
      <esi:include src="http://www.myplace.com/r4c5/ads/news1.html"/>
      </esi:attempt>
      <esi:except>
      This spot is reserved for your company's advertising. For more
      information, <a href="http://www.biz.myplace.com/adpolicies/
      columnads.html"> click here!</a>
      </esi:except>
</esi:try>
</TD></TR>
<esi:comment text="NOTE -- this ENDS content row. Other rows repeat this
      construction but with News becoming Sports, Music, etc.">


</TABLE>
</BODY>
</HTML>
```



Index

Symbols

A

B

C