



# Configuring Sticky Parameters for Content Rules

---

This chapter describes how to configure sticky parameters for content rules. The information in this chapter applies to all CSS models, except where noted. This chapter contains the following sections:

- Sticky Overview
- Configuring Sticky on the CSS
- Specifying an Advanced Load Balance Method for Sticky Content
- Configuring Sticky Serverdown Failover
- Configuring Sticky Mask
- Configuring Sticky Inactive Timeout
- Configuring Sticky Content for SSL
- Configuring String Range
- Specifying a String Operation
- Enabling or Disabling String ASCII Conversion
- Specifying End of String Characters
- Specifying a String Prefix
- Specifying a String Process Length
- Specifying a String Skip Length
- Showing Sticky Configurations
- Configuring Sticky Parameters for E-Commerce Applications

# Sticky Overview

During a session, the CSS maintains an association between a client and a server. This association is referred to as *stickiness*. Stickiness enables transactions over the Web when the a client must remain on the same server for the entire session. Depending on the content rule, the CSS “sticks” a client to an appropriate server after the CSS has determined which load balancing method to use.

If the CSS determines that a client is already stuck to a particular service, then the CSS places the client request on that service, regardless of the load balancing criteria specified by the matched content rule. If the CSS determines that the client is not stuck to a particular service, it applies normal load balancing to the content request.

Client *cookies* uniquely identify clients to the services providing content. A cookie is a small data structure used by a server to deliver data to a Web client and request that the client store the information. In certain applications, the client returns the information to the server to maintain the state between the client and the server.

When the CSS examines a request for content and determines through content rule matching that the content is sticky, it examines any cookie or URL present in the content request. The CSS uses this information to place the content request on the appropriate server.

This section describes the following topics:

- Why Use Stickiness?
- Using Layer 3 Sticky
- Using Layer 4 Sticky

## Why Use Stickiness?

When customers visit an e-commerce site, they usually start out by browsing around the site, the Internet equivalent of window-shopping. Depending on the application, the site may require that the customer become “stuck” to one server once the connection is established, or the application may not require this until the customer starts to build a shopping cart.

In either case, once the customer adds items to the shopping cart, it is important that all of the customer's requests get directed to the same server so that all the items are contained in one shopping cart on one server. An instance of a customer's shopping cart is typically local to a particular Web server and is not duplicated across multiple servers.

E-commerce applications are not the only types of applications that require stickiness. Any Web application that maintains client information may require stickiness, such as banking applications or online trading.

Because the application must distinguish each user or group of users, the CSS needs to determine how a particular user is stuck to a specific Web server. The CSS can use a variety of methods, including:

- Source IP address
- Source IP address and destination port
- String found in a cookie or a URL
- SSL session ID

The e-commerce application itself dictates which of these methods is appropriate for a particular e-commerce vendor.

## Using Layer 3 Sticky

If an application requires that a user be stuck for the entire session, use Layer 3 sticky, which sticks a user to a server based on the user's IP address. The CSS has a sticky table of 32K, which means that once 32K simultaneous users are on the site, the table wraps and the first users become “unstuck”. If the volume of your site is such that you will have more than 32K users at a time, or if a large percentage of your customers come to you through a mega-proxy, then consider using either a different sticky method (for example, the advanced-balance **cookie**, **cookieurl**, or **url**), or increasing your sticky mask.

The default sticky mask is 255.255.255.255, which means that each entry in the sticky table is an individual IP address. Some mega-proxies allow one user to use several different IP addresses in a range of addresses over the life of one session. This causes some of the TCP connections to get stuck to one server, and other TCP connections to a different server for the same transaction. This would result in possibly losing some items from the shopping cart. To avoid this problem, use one of the more advanced methods of sticking. If you cannot, Cisco recommends using a sticky mask of 255.255.240.0.

## Using Layer 4 Sticky

Layer 4 sticky functions identically to Layer 3 sticky, except that it sticks based on the source IP address and destination port combination. Layer 4 sticky also uses the sticky table and has the same limitations as Layer 3 sticky as well as SSL, because if the CSS sees the same IP address with two different destination ports, it will use two entries. You can also apply sticky mask to Layer 4 sticky.

If you are concerned about whether your site can handle all of the simultaneous sessions, then consider using the advanced-balanced methods of **cookie**, **cookieurl**, or **url**.

## Configuring Sticky on the CSS

Configuring sticky on the CSS requires you to:

- Determine the sticky method you want to use according to the requirements of the site (for example, Layer 3, Layer 4, or one of the string methods).
- Configure a failover method.

If you use advanced-balance methods **cookies**, **url**, or **cookieurl**, you must also:

- Determine whether you want to use an exact string match or a hash, and then configure that function.
- Determine how you want to delimit (configure) the string.

To configure sticky on the CSS:

1. Configure the sticky method using the **advanced-balance** command and its options. The **advanced-balance** command options are described in “Specifying an Advanced Load Balance Method for Sticky Content” later in this chapter.
  - To configure Layer 3 sticky, use **advanced-balance sticky-srcip** in the content rule. If necessary, change the sticky mask from the default of 255.255.255.255.
  - To configure Layer 4 sticky, use **advanced-balance sticky-srcip-dstport** in the content rule. If necessary, change the sticky mask from the default of 255.255.255.255.
  - To configure sticky cookies, use **advanced-balance cookies** in the content rule.

- To configure sticky URL, use **advanced-balance url** in the content rule.
  - To configure sticky cookies with URLs, use **advanced-balance cookieurl** in the content rule.
2. Configure a failover method. Use the **sticky-serverdown-failover** command to define what will happen if a sticky string is found, but the associated service has failed or is suspended. The sticky failover default is for the CSS to use the configured load balancing method. The **sticky-serverdown-failover** options are described in the section “Configuring Sticky Serverdown Failover” later in this chapter.

If you configured an advanced-balance method of **sticky-srcip** or **sticky-srcip-dstport**, no further steps are required.

If you configured the advanced-balance methods **cookies**, **url**, or **cookieurl**, complete Steps 3 and 4.

3. If you are using **advanced-balance cookies**, **url**, or **cookieurl**, determine whether you want to use an exact string match or a hash.

To use an exact string match:

- a. Enter the **string operation match-service-cookie** command (this is the default for the **string operation** command).
- b. For each service configuration, enter the service mode **string** command to configure the unique string that you want to use for matching each server.

For example, you have three servers and you want the string matching to be *serverid111* for service1, *serverid112* for service2, and *serverid113* for service3. Configure the Web server applications to use these strings when they set cookies or pass parameters.

For information on the **string operation match-service-cookie** command, refer to the section “Specifying a String Operation” later in this chapter.

To use the hash algorithm:

- a. Enter the **string operation** command in the content rule.
- b. Select an option (**hash-a**, **hash-crc32**, or **hash-xor**) depending on the hash method you wish to use. Hashing requires that each server can accept cookies set by all other servers.

Technical Support recommends using either **hash-xor** or **hash-crc32**, depending on your string possibilities. If the strings are completely dissimilar, use **hash-xor**. If the strings are similar, use **hash-crc32**. For example, if your string values are *abc1*, *abc2*, and *abc3*, the **hash-xor** method cannot provide you with enough variance in the hash values (that is, *abc1* and *abc2* may end up on the same server because they may hash to the same value).

For information on the string operation hash options, refer to the section “Specifying a String Operation” later in this chapter.

4. If you are using **advanced-balance cookie**, **url**, or **cookieurl**, determine how you want to delimit (configure) the string. Use the following owner-content **string** commands to delimit the string:
  - **string range** - Defining the string range enables you to limit the size of the search. By default the CSS searches the first 100 bytes of the cookie, URL, or parameters in the URL depending on the method. If you know where in the cookie or URL the string is likely to appear, define the string range accordingly. The range can be from 1 to 1000 (default is 1 to 100). The string range options are described in the section “Configuring String Range” later in this chapter.
  - **string eos-char** - A maximum of 3 ASCII characters that delimit the end of the string. Use this option when the string length varies. Note that **string process-length** overrides **string eos-char**. If you do not configure either option, the CSS uses a maximum of 100 bytes for the delimiter.
  - **string prefix** - The CSS uses the string prefix (maximum of 30 characters) to locate the string within the cookie or URL. If the string prefix is specified, but not found, the CSS uses the normal balance method.
  - **string process-length** - Specifies the number of bytes after the end of the prefix plus the skip-length that is used to determine the string. Use this option when the string length is fixed.
  - **string skip-length** - Specifies the number of bytes to skip after the end of the prefix. The range is 0 to 64.

For example, if you are using `ipaddr=192.168.3.6&`, then use the **string prefix** `"ipaddr="` and the **string eos-char** `"&"` because the IP addresses vary in length.

For example, if you are using server ID=*server111*, then use the **string prefix** “server ID=” and a **string process-length** of 8 because the string length does not vary in length.

Table 1-1 describes sticky rules and how they apply to content rules.

**Table 1-1 Applying Sticky Rules to Content Rules**

<b>Rule Type</b>	<b>Sticky Configuration</b>	<b>Stickiness Based on...</b>
Layer 3 content rule	<b>advanced-balance sticky-srcip</b>	Source IP address using a sticky mask.
Layer 4 content rule	<b>advanced-balance sticky-srcip-dstport</b>	Source IP address and destination port using a sticky mask.
Layer 5 content rule not using a sticky string	<b>advanced-balance sticky-srcip-dstport</b>	Source IP address and destination port using a sticky mask.
Layer 5 content rule using a sticky string	<b>advanced-balance cookies</b> or <b>advanced-balance cookieurl</b>	Searching for a sticky string in the cookie or URL. If the CSS does not find the sticky string in the cookie or URL, the CSS load-balances each request among the available servers.
Layer 5 content rule with SSL	<b>advanced-balance ssl</b>	SSL v3 session ID. If no session ID is present, the CSS uses the source IP address and destination port to maintain stickiness.

# Specifying an Advanced Load Balance Method for Sticky Content

Use the **advanced-balance** command to specify an advanced load-balancing method for a content rule that includes stickiness. A content rule is “sticky” when additional sessions from the same user or client are sent to the same service as the first connection, overriding normal load balancing. By default, the advanced balancing method is disabled.

The advanced-balance command options **cookie**, **cookieurl**, and **url** use strings for sticking clients to servers. These options are beneficial when the sticky table limit is too small for your application requirements because the string methods do not use the sticky table.

The syntax and options for this content mode command are:

- **advanced-balance arrowpoint-cookie** - Enable the content rule to stick the client to the server based on the unique service identifier information of the selected server in the ArrowPoint-generated cookie. Configure the service identifier by using the **(config-service) string** command. For information on configuring the ArrowPoint-generated cookie, refer to the **(config-owner-content) arrowpoint-cookie** command. You can use this option with any Layer 5 content rule.



---

**Note**

If you are using the **arrowpoint-cookie** option of the **advanced-balance** command (refer to “Configuring Sticky Parameters for E-Commerce Applications” later in this chapter), do not configure string match criteria, the **sticky-no-cookie-found-action** command, or the **sticky-serverdown-failover** command.

---

- **advanced-balance cookies** - Enable the content rule to stick the client to the server based on the configured string found in the HTTP cookie header. You must specify a port in the content rule to use this option. The CSS will then spoof the connection. A content rule with a sticky configuration set to **advanced-balance cookies** requires all clients to enable cookies on their browser.

When a client makes an initial request, they do not have a cookie. But once they go to a server that is capable of setting cookies, they receive the cookie from the server. Each subsequent request contains the cookie until the cookie expires. A string in a cookie can be used to stick a client to a server. The service mode **string** command enables you to specify where the CSS should locate the string within the cookie.

The CSS processes the cookie using:

- An exact match that you set up when you configure the services.
- Data for a hash algorithm. For more information, refer to “Comparing Hash Method With Match Method” later in this chapter.
- **advanced-balance cookieurl** - Same as advanced-balance cookies, but if the CSS cannot find the cookie header in the HTTP packet, this type fails over to look up the URL extensions (that is, the portion after the "?" in the URL) based on the same string criteria. You must specify a port in the content rule to use this option. The CSS will then spoof the connection.

This option is useful if a Microsoft<sup>®</sup> IIS web server is used with Cookie Munger, which dynamically places the session state information in the cookie header or URL extension, depending on whether or not the client can accept cookies.

Some client applications do not accept cookies. When a site depends upon the information in the cookie, administrators sometimes modify the server application so that it appends the cookie data to the parameters section of the URL. The parameters typically follow a "?" at the end of the main data section of the URL.

The **advanced-balance cookieurl** command sticks a client to a server based on locating the configured string:

- In the cookie if a cookie exists
- In the parameters section of the URL if no cookie exists

The string can either be an exact match or be hashed.

- **advanced-balance none** - Disable the advanced-balancing method for a content rule (default).
- **advanced-balance sticky-srcip** - Enable the content rule to stick a client to a server based on the client IP address, also known as Layer 3 stickiness. You can use this option with Layer 3, 4, or 5 content rules.

- **advanced-balance sticky-srcip-dstport** - Enable the content rule to stick a client to a server based on both the client IP address and the server destination port number, also known as Layer 4 stickiness. You can use this option with Layer 4 or Layer 5 content rules.
- **advanced-balance ssl** - Enable the content rule to stick the client to the server based on the Secure Socket Layer (SSL) version 3 session ID assigned by the server. The application type must be SSL for the content rule. You must specify a port in the content rule to use this option. The CSS will then spoof the connection.

Sites where encryption is required for security purposes often use SSL. SSL contains session IDs and the CSS has the ability to use these session IDs for sticking the client to a server. In order for the CSS to successfully provide SSL stickiness, the application must be using SSL version 3 session IDs. Sticky SSL uses the sticky table. If you are concerned about the number of concurrent sessions, and not concerned about security, you should consider using **cookie**, **cookieurl**, or **url**.

- **advanced-balance url** - Enable the content rule to stick a client to a server based on a configured string found in the URL of the HTTP request. You must specify a port in the content rule to use this option. The CSS will then spoof the connection.

Similar to **advanced-balance cookie**, **advanced-balance url** may use either an exact match method or a hash algorithm. The string can exist anywhere in the URL.

- **advanced-balance wap-msisdn** - Enable a Layer 5 content rule to stick a client to a server based on the MSISDN header field in an HTTP request. MSISDN is the header field for wireless clients using the Wireless Application Protocol (WAP). The MSISDN field value can contain the client's telephone number or user ID, which uniquely identifies a client. This command is especially useful for clients using e-commerce applications, for example, making a purchase on a web site.

If the MSISDN header is present in an HTTP request, the CSS generates a key based on the value in the MSISDN header field. The CSS uses the key to look up an entry in the sticky table. If an entry exists in the sticky table, the CSS sends the client to the sticky server indicated by the table entry. If an entry does not exist in the sticky table, the CSS:

- Generates a new entry in the sticky table (similar to L3, L4, and SSL sticky).

- Load balances the request to a server.
- Stores the selected server and the key (hashed value of the MSISDN header) in the sticky entry.

For subsequent requests from the same client, the CSS looks up the same table entry and sends the client to the same server.

If the MSISDN header field is not present in an HTTP request, the CSS load balances the client request based on the configured balance method. The default load-balancing method is roundrobin.

In the following example, TCP port 80 traffic destined for 192.168.128.151 is stuck to either server1 or server2 based on the contents of the MSISDN HTTP header field.

```
owner arrowpoint
content ruleWapSticky
  vip address 192.168.128.151
  protocol tcp
  port 80
  url "/*"
  add service server1
  add service server2
  advanced-balance wap-msisdn
  active
```

For example, to specify **advanced-balance wap-msisdn** for content rule *rule1*, enter:

```
(config-owner-content[arrowpoint-rule1])# advanced-balance
wap-msisdn
```

**Note**

You can use the **advanced-balance wap-msisdn** command alone or with the MSISDN header field type. For a configuration example using both, refer to “Configuring Wireless Users for E-Commerce Applications” later in this chapter.

To disable the advanced load-balancing method, enter:

```
(config-owner-content[arrowpoint-rule1])# advanced-balance none
```

# Configuring Sticky Serverdown Failover

Use the **sticky-serverdown-failover** command to define what will happen if a sticky string is found, but the associated service has failed or is suspended. The sticky failover default method is for the CSS to use the configured load-balancing method.



## Note

If you are using the **arrowpoint-cookie** option of the **advanced-balance** command (refer to “Configuring Sticky Parameters for E-Commerce Applications” later in this chapter), do not configure string match criteria, the **sticky-no-cookie-found-action** command, or the **sticky-serverdown-failover** command.

The syntax and options for this content mode command are:

- **sticky-serverdown-failover balance** - Set the failover method to use a service based on the configured load-balancing method.
- **sticky-serverdown-failover redirect** - Set the failover method to use the redirect string configured on a content rule. If you do not configure a redirect string on a content rule, the load-balancing method is used.
- **sticky-serverdown-failover reject** - Reject the content request.
- **sticky-serverdown-failover sticky-srcip** - Set the failover method to use a service based on the client source IP address.
- **sticky-serverdown-failover sticky-srcip-dstport** - Set the failover method to use a service based on the client source IP address and the server destination port.

For example, to set the sticky failover method to **sticky-srcip**, enter:

```
(config-owner-content[arrowpoint-rule1]) sticky-serverdown-failover
sticky-srcip
```

To set the sticky failover method to its default setting of using the configured load-balancing method, enter:

```
(config-owner-content[arrowpoint-rule1])# no
sticky-serverdown-failover
```

## Configuring Sticky Mask

A client IP address uniquely identifies the client to the CSS. During normal client-server sessions, the IP address is maintained throughout the connection. However, if the connection is lost (for example, due to a dense proxy failover) and the client reconnects with a different IP address, the CSS needs to reconnect the client to the same server that is preserving the client information (for example, information from a shopping cart or financial session).

Use the **sticky-mask** command to mask a group of client IP addresses in order to preserve the client connection state when the client's source IP address changes. The sticky mask specifies which portion of the client IP address the CSS will mask. The default sticky subnet mask is 255.255.255.255.

For example:

```
(config-owner-content[arrowpoint-rule1])# sticky-mask  
255.255.255.0
```

To restore the sticky subnet mask to the default of 255.255.255.255, enter:

```
(config-owner-content[arrowpoint-rule1])# no sticky-mask
```

## Configuring Sticky Inactive Timeout

Use the **sticky-inact-timeout** command to specify the inactivity timeout period on a sticky connection for a content rule before the CSS removes the sticky entry from the sticky table. When you configure this period, the CSS keeps the sticky entry in the sticky table for the specified amount of time. The CSS does not reuse this entry until the time expires. If the sticky table is full and none of the entries have expired, the CSS rejects the new sticky request. When the sticky connection expires, the CSS uses the configured load-balancing method to choose an available server for the request.

When this feature is disabled, the new sticky connection uses the oldest used sticky entry. A sticky association could exist for a time depending on the sticky traffic load on the CSS.

The syntax for this command is:

```
sticky-inact-timeout minutes
```

Enter the number of minutes of inactivity from 0 to 65535. The default value is 0, which means this feature is disabled. For example:

```
(config-owner-content[arrowpoint-rule1])# sticky-inact-timeout 9
```

To disable the sticky connection inactivity timeout feature, enter:

```
(config-owner-content[arrowpoint-rule1])# no sticky-inact-timeout
```

## Configuring Sticky Content for SSL

To use stickiness based on SSL version 3 session ID, configure a specific SSL Layer 5 rule for a service. To configure an SSL Layer 5 rule for a service:

- Set the port to 443 using the **(config-owner-content) port** command.
- Set the URL to “/\*” using the **(config-owner-content) url** command.
- Enable the content rule to be sticky based on SSL using the **(config-owner-content) advanced-balance ssl** command.
- Specify the SSL application type using the **(config-owner-content) application ssl** command.

For example, the following owner portion of a startup-config shows a content rule configured for SSL.

```
!***** OWNER *****!
owner arrowpoint
content L5sslsticky
vip address 192.3.6.58
add service server87
add service server88
balance aca
protocol tcp
port 443
url "/*"
advanced-balance ssl
application ssl
active
```

# Configuring String Range

Use the **string-range** command to specify the starting and ending byte positions within a cookie, URL, or URL extension the CSS uses to search for the specified string. By specifying this range of bytes, the CSS processes the information located only within this range. This limits the amount of information that the CSS has to process when examining each cookie, URL, or URL extension, enhancing its performance.



## Note

---

If the starting position is beyond the cookie, URL or URL extension, the CSS does not perform the string function. When the ending position is beyond the cookie, URL, or URL extension, the string processing stops at the end of the corresponding header.

---

Enter the *start\_byte* as the starting byte position of the cookie, URL, or URL extension after the header. Enter an integer from 1 to 1999. The default is 1. Ensure that the starting byte position is less than the end byte.

Enter the *end\_byte* as the ending byte position of the cookie, URL, or URL extension. Enter an integer from 2 to 2000. The default is 100. Ensure that the ending byte position is more than the start byte position.

If you are using advanced-balance:

- **cookie**, the CSS starts counting after "Cookie: "(that is, cookie, colon, space).
- **url**, the CSS starts counting after the "/".
- **cookieurl**, the CSS starts counting after the "Cookie: " string. If the CSS does not find "Cookie: " in the HTTP request, it starts counting after the "?" in the URL of the same request.

For example:

```
(config-owner-content[arrowpoint-rule1])# string-range 35  
to 55
```

To restore the string range to the default of 1 to 100, enter:

```
(config-owner-content[arrowpoint-rule1])# no string-range
```

# Specifying a String Operation

Use the **string operation** command to determine the method to choose a destination server for a string result. The CSS derives the string result from the settings of the **string** criteria commands. You can choose a server by using the configured balance method or by using the hash key generated by the specified sticky hash type. If the Web servers:

- Are only capable of accepting the cookies that they set, then you must use the exact match method.
- Can accept any cookies that are set by either a cookie server or other servers, then you may use the hash method.

**Note**

---

If you are using the **arrowpoint-cookie** option of the **advanced-balance** command (refer to “Configuring Sticky Parameters for E-Commerce Applications” later in this chapter), do not configure string match criteria, the **sticky-no-cookie-found-action** command, or the **sticky-serverdown-failover** command.

---

## Comparing Hash Method With Match Method

When an application uses the exact match method, once a client makes a request to a particular server, the server is responsible for providing the client with a string unique to the server to use for future requests. Typically, if a server receives a string in a request that was set by another server, that string causes an error. In an exact match, the CSS looks for the unique string. If it finds an exact match, then the server is used. If no match is found, the CSS uses the configured load-balancing method to select a server for the client.

When an application uses one of the hash algorithms, all of the servers are capable of accepting any strings set by other servers. The model was designed so you could set up a site where the initial login would send a client to a Web server that assigns cookies to clients. When the CSS receives the first request from a client with the cookie string, it performs the hash operation on the string and chooses a server accordingly. The hash algorithm ensures that a particular string is always sent to a specific server, but it does not have to be a predefined server, as with an exact match.

Using the string operation hash algorithms may allow the Web server application to be used without being modified. When you use the **string operation match-service-cookie** method, you must modify the Web server application so that each server generates a unique string. The hash algorithms may be able to take advantage of strings already generated by the servers.

The syntax and options for this content mode command are:

- **string operation match-service-cookie** - Choose a server by matching a service cookie in the sticky string. This is the default setting. When a match is not found, the CSS chooses the server by using the configured balance method (for example, roundrobin).
- **string operation [hash-algorithm|hash-crc32|hash-xor]** - Choose a server by using the hash key generated by the designated hash method. When using advanced balance cookies with a hash algorithm, all servers in the same domain must accept cookies regardless of which server created the cookie. This enables all servers configured on the Layer 5 rule to process cookies passed in an HTTP request.
  - **hash-a** - Apply a basic hash algorithm on the hash string to generate the hash key
  - **hash-crc32** - Apply the CRC32 algorithm on the hash string to generate a hash key
  - **hash-xor** - Exclusive OR (XOR) each byte of the hash string to derive the final hash key

If the selected server is out of service, the CSS performs a rehash to choose another server.

Technical Support recommends using either **hash-xor** or **hash-crc32** depending on your string possibilities. If the strings are completely dissimilar, use **hash-xor**. If the strings are similar, use **hash-crc32**. For example, if your string values are *abc1*, *abc2*, and *abc3*, the **hash-xor** method cannot provide you with enough variance in the hash values (that is, *abc1* and *abc2* may end up on the same server because they may hash to the same value).

For example, to set the string operation to choose a server by using the string operation **hash-crc32** algorithm, enter:

```
(config-owner-content[arrowpoint-rule1])# string operation  
hash-crc32
```

To reset the string operation to its default setting of choosing a server by matching a service cookie in the sticky string, enter:

```
(config-owner-content[arrowpoint-rule1])# no string operation
```

The CSS derives a string result from the following string criteria commands:

- **string ascii-conversion**
- **string eos-char**
- **string prefix**
- **string process-length**
- **string skip-length**

## Enabling or Disabling String ASCII Conversion

Use the **string ascii-conversion** command to enable or disable the ASCII conversion of escaped special characters within the specified sticky string range before applying any processing to the string. By default, ASCII conversion is enabled.

For example, to disable ASCII conversion of escaped special characters, enter:

```
(config-owner-content[arrowpoint-rule1])# string ascii-conversion  
disable
```

To reenable the ASCII conversion of escaped special characters to its default setting, use the **no** form of the command or the **enable** option. For example:

```
(config-owner-content[arrowpoint-rule1])# no string  
ascii-conversion
```

```
(config-owner-content[arrowpoint-rule1])# string ascii-conversion  
enable
```

## Specifying End of String Characters

Use the **string eos-char** command to specify up to three ASCII characters as the delimiters for the sticky string. For example, in a cookie header, a semicolon (;) character is usually used as a delimiter; in a URL extension, an ampersand (&) character is often used as a delimiter.

The CSS uses the **string eos-char** value if the **(config-owner-content) string process-length** command is not configured. The **(config-owner-content) string process-length** command has higher precedence. If neither command is configured, the CSS uses the maximum of 100 bytes for the final string length. Enter the sticky string end of string characters as a quoted text string with a maximum of three characters.

For example:

```
(config-owner-content[arrowpoint-rule1])# string eos-char “;”
```

To clear the end of string characters, enter:

```
(config-owner-content[arrowpoint-rule1])# no string eos-char
```

## Specifying a String Prefix

Use the **string prefix** command to specify the string prefix located in the string range. If you do not configure the string prefix, the string functions start from the beginning of the cookie, URL, or URL extension, depending on the sticky type. If the string prefix is configured but is not found in the specified string range, the CSS uses the load-balancing method you defined in the **sticky-serverdown-failover** command. Enter the string prefix as a quoted text string with a maximum of 30 characters. The default is no prefix (“”).

For example:

```
(config-owner-content[arrowpoint-rule1])# string prefix “UID=”
```

To clear the string prefix, enter:

```
(config-owner-content[arrowpoint-rule1])# no string prefix
```

## Specifying a String Process Length

Use the **string process-length** command to specify how many bytes, after the end of prefix designated by the **string prefix** command and skipping the bytes designated by the **string skip-length** command, the string action will use. This command has higher precedence than the **string eos-char** command. If neither command is configured, the CSS uses the maximum of 100 bytes for the final string action. Enter the number of bytes from 0 to 64. The default is 0.

For example:

```
(config-owner-content[arrowpoint-rule1])# string process-length 16
```

To set the number of bytes to its default setting of 0, enter:

```
(config-owner-content[arrowpoint-rule1])# no string process-length
```

## Specifying a String Skip Length

Use the **string skip-length** command to specify how many bytes to skip after the end of prefix to find the string result. Enter the number of bytes from 0 to 64. The default is 0. For example:

```
(config-owner-content[arrowpoint-rule1])# string skip-length 3
```

To set the number of bytes to its default setting of 0, enter:

```
(config-owner-content[arrowpoint-rule1])# no string skip-length
```

## Configuring Sticky-No-Cookie-Found-Action



### Note

---

If you intend to use the **advanced-balance arrowpoint-cookie** command (refer to “Configuring Sticky Parameters for E-Commerce Applications” later in this chapter), do not configure the **sticky-no-cookie-found-action** command.

---

Use the **sticky-no-cookie-found-action** command to specify the action the CSS should take for a sticky cookie content rule when it cannot locate the cookie header or the specified cookie string in the client request. The syntax for this content mode command is:

```
sticky-no-cookie-found-action
```

The options are:

- **loadbalance** (default) - The CSS uses the configured balance method when no cookie is found in the client request.

- **redirect** “*redirect URL*” - The CSS redirects the client request to a specified URL string when no cookie found in the client request. When using this option, you must also specify a redirect URL. Specify the redirect URL as a quoted text string from 0 to 64 characters.
- **reject** - The CSS rejects the client request when no cookie is found in the request.
- **service** *service\_name* - The CSS sends the no cookie client request to the specified service when no cookie is found in the request.

For example:

```
(config-owner-content[arrowpoint-rule1])#  
sticky-no-cookie-found-action redirect  
"http://www.lml.com/nocookie.html"
```

To reset sticky-no-cookie-found-action to the default of loadbalance, enter:

```
(config-owner-content[arrowpoint-rule1])# no  
sticky-no-cookie-found-action
```

## Showing Sticky Configurations

To display sticky configurations for content, enter the **show rule** command. For example:

```
(config-owner-content[arrowpoint-rule1])# show rule
```

For details on the **show rule** command, refer to the *Content Services Switch Basic Configuration Guide*, Chapter 7, Configuring Content Rules.

# Configuring Sticky Parameters for E-Commerce Applications

By configuring sticky parameters for e-commerce applications, you can instruct the CSS how to process client requests that do not contain cookies when the requests are destined to a content rule that is sticking based on a string within a cookie. You can also instruct the CSS how to process wireless users by integrating HTTP header load balancing with the **advanced-balance wap-msisdn** command. For applications that use the CSS sticky table, you can remove a sticky table entry after a defined period of activity.

Configuring sticky parameters for e-commerce applications includes:

- Configuring an Advanced Balance ArrowPoint Cookie
- Configuring an ArrowPoint Cookie
- Configuring an Arrowpoint-Cookie Expiration Time
- Configuring an Arrowpoint Cookie Path
- Configuring Sticky-No-Cookie-Found-Action
- Configuring Wireless Users for E-Commerce Applications

## Configuring an Advanced Balance ArrowPoint Cookie



### Note

---

If you are using the **arrowpoint-cookie** option of the **advanced-balance** command, do not configure string match criteria, the **sticky-no-cookie-found-action** command, or the **sticky-serverdown-failover** command.

---

Use the **advanced-balance arrowpoint-cookie** command to enable the content rule to stick the client to the server based on the unique service identifier information of the selected server in the ArrowPoint-generated cookie. Configure the service identifier by using the **(config-service) string** command. You do not need to configure string match criteria. For information on configuring the ArrowPoint-generated cookie, refer to the **(config-owner-content) arrowpoint-cookie** command. You can use this option with any Layer 5 content rule.

For example, to specify **advanced-balance arrowpoint-cookie** for content *rule1*, enter:

```
(config-owner-content[arrowpoint-rule1])# advanced-balance  
arrowpoint-cookie
```

To disable the advanced load-balancing method, enter:

```
(config-owner-content[arrowpoint-rule1])# no advanced-balance
```

## Configuring an ArrowPoint Cookie

Use the **arrowpoint-cookie** command to configure the ArrowPoint cookie path and expiration time. The CSS generates the ArrowPoint cookie transparently for a client, the client stores it and returns it in subsequent requests, and the CSS later uses it to maintain the client-server stickiness. This cookie contains the sticky information itself and does not refer to a sticky table.

If you configure the **arrowpoint-cookie** method in a content rule, the CSS always checks for the existence of the ArrowPoint cookie when it receives a client request. If this cookie does not exist, the CSS performs server load balancing and generates an ArrowPoint cookie.

When the CSS finds the cookie in the client request, it unscrambles the cookie data and then validates it. Then, the CSS checks the cookie expiration time. If the cookie has expired, the CSS sends a new cookie containing the information about the server where the client was stuck. This appears as an uninterrupted connection.

If the cookie format is valid, the CSS ensures the consistency between the cookie and the CSS configuration. When all the validations are passed, the CSS forwards the client request to the server indicated by the server identifier. Otherwise, the CSS treats this request as an initial request.

The options for this content mode command are:

- **arrowpoint-cookie expiration** - Set an expiration time, which the CSS compares with the time associated with the cookie
- **arrowpoint-cookie path** - Set the cookie path to a configured path
- **arrowpoint-cookie browser-expire** - Allow the browser to expire the cookie
- **arrowpoint-cookie expire services** - Expire the service information when the cookie expires

## Configuring an Arrowpoint-Cookie Expiration Time

Use the **arrowpoint-cookie expiration** command to set an expiration time, which the CSS compares with the time associated with the ArrowPoint cookie. If the cookie has expired, the CSS sends a new cookie that includes the server where the client was stuck. This allows for the appearance of an uninterrupted connection. If you do not set an expiration time, the cookie expires when the client exits the browser.

The syntax of this owner-content mode configuration command is:

**arrowpoint-cookie expiration *dd:hh:mm:ss***

The variables are:

- *dd* - Number of days. Valid numbers are from 00 to 99.
- *hh* - Number of hours. Valid numbers are from 00 to 99.
- *mm* - Number of minutes. Valid numbers are from 00 to 99.
- *ss* - Number of seconds. Valid numbers are from 00 to 99.



### Note

Do not use all zeros for days, hours, minutes, and seconds. This value is invalid.

For example:

```
(config-owner-content[arrowpoint-rule1])# arrowpoint-cookie  
expiration 08:04:03:06
```

To reset the expiration time to when the client exits the browser, enter:

```
(config-owner-content[arrowpoint-rule1])# no arrowpoint-cookie  
expiration
```

## Configuring Arrowpoint-Cookie Browser Expire

Use the **arrowpoint-cookie browser-expire** command to allow the browser to expire the ArrowPoint cookie based on the expiration time. To configure the expiration time, refer to the **arrowpoint-cookie expiration** command earlier in this chapter. The syntax of this owner-content configuration mode command is:

```
arrowpoint-cookie browser-expire
```

For example:

```
(config-owner-content[arrowpoint-rule1])# arrowpoint-cookie  
browser-expire
```

To allow the CSS to expire the cookie, enter:

```
(config-owner-content[arrowpoint-rule1])# no arrowpoint-cookie  
browser-expire
```

**Note**

---

When the cookie expires, all sticky information is lost.

---

## Configuring Arrowpoint-Cookie Expire Services

Use the **arrowpoint-cookie expire-services** command to expire service information when the cookie expires before sending a new cookie. By default, when the cookie expires, the CSS sends a new cookie with the server information from the expired cookie. The syntax of this owner-content configuration mode command is:

```
arrowpoint-cookie expire-services
```

For example:

```
(config-owner-content[arrowpoint-rule1])# arrowpoint-cookie  
expire-services
```

To reset the default behavior, enter:

```
(config-owner-content[arrowpoint-rule1])# no arrowpoint-cookie  
expire-services
```

## Configuring an Arrowpoint Cookie Path

Use the **arrowpoint-cookie path** command to set the ArrowPoint cookie path to a configured path. Otherwise, the CSS sets the default path attribute of the cookie to `/`. The syntax for this owner-content configuration mode command is:

```
arrowpoint-cookie path "path_name"
```

Enter the *path\_name* where you want to send the cookie. Enter a quoted text string with a maximum of 99 characters. The default path of the cookie is `/`.

For example:

```
(config-owner-content[arrowpoint-rule1])# arrowpoint-cookie path  
"/cgi-bin/"
```

To reset the cookie path to its default of `/`, enter:

```
(config-owner-content[arrowpoint-rule1])# no arrowpoint-cookie  
path
```

## Configuring Wireless Users for E-Commerce Applications

Use the **advanced-balance wap-msisdn** command with the MSISDN header field to configure wireless users for e-commerce applications. For details on the **advanced-balance wap-msisdn** command, refer to “Specifying an Advanced Load Balance Method for Sticky Content” earlier in this chapter. For details on the MSISDN header field, refer to Chapter 2, Configuring HTTP Header Load Balancing, in the section “Configuring a Header Field Entry”.

Wireless clients use the wireless application protocol (WAP) to access Internet content. When a wireless client sends a request for content, the WAP protocol gateway (a device that translates requests from the WAP protocol stack to the WWW protocol stack) generates the MSISDN field and adds it to the HTTP header.

In the following example, TCP port 80 traffic destined for VIP 192.168.128.151 that contains the string “012” in the MSISDN HTTP header field will hit content rule rule012. The CSS will stick this traffic to either server1 or server2 based on the entire contents of the MSISDN field.

TCP port 80 traffic destined for 192.168.128.151 that does not contain the string "012" in the MSISDN HTTP header field, but has the field in the header, will hit content rule ruleNo012. The CSS will roundrobin load-balance the traffic across server21 and server22.

TCP port 80 traffic destined for 192.168.128.151 that does not contain the MSISDN HTTP header field will hit content rule ruleNoWap. The CSS will roundrobin load-balance the traffic across server31 and server32.

```
header-field-group wap012
  header-field 1 wap-msisdn contain "012"

header-field-group wapNo012
  header-field 1 wap-msisdn not-contain "012"

owner arrowpoint
content rule012
  vip address 192.168.128.151
  protocol tcp
  port 80
  url "/*"
  add service server1
  add service server2
  header-field-rule wap012
  advanced-balance wap-msisdn
  active

content ruleNo012
  vip address 192.168.128.151
  protocol tcp
  port 80
  url "/*"
  add service server21
  add service server22
  header-field-rule wapNo012
  active

content ruleNoWap
  vip address 192.168.128.151
  protocol tcp
  port 80
  url "/*"
  add service server31
  add service server32
  active
```

