WebAccelerator Policy Management Guide

version 5.1

MAN-0210-00

Service and Support Information

Product Version

This manual applies to product version 5.1 of the WebAccelerator™.

Legal Notices

Patents

Table of Contents

Preface xi
Related Information xii
Conventions Used in this Book xii

Chapter 1 Introduction 1
WebAccelerator Components 2
Managing Accelerators 3
Serving and Caching Pages 3
Managing Content 4
Customizing WebAccelerator Behavior 6
    Application Matching Policies 7
    Application Caching Policies 9
Performance Reports 10
ESI Support 11

Chapter 2 Using the Administration Tool 13
Accessing the Admin Tool 14
Navigating the User Interface 14
    Common Tasks 15
    Preferences 17
    Help 17
Starting at the Home Page 18

Chapter 3 Managing Applications 19
WebAccelerator Clusters 20
Application Profiles 20
Host Maps 21
    Unmapped Hosts 21
    Valid Hostnames 22
Request Handling 23
Creating a Cluster 25
Creating an Application Profile 28
    Invalidation Service Password 31
Defining Destination Hosts 32
Managing Express Connect 34

Chapter 4 Managing Policies 35
What Is a Policy Set? 36
Managing Policy Sets 37
Creating Policy Sets 38
Copying Policy Sets 38
Renaming Policy Sets 38
Deleting Policy Sets 39
Exporting Policy Sets 39
Importing Policy Sets 39
Importing and Exporting Transforms 40
Using the Policy Editor 40
Policy Editor Basics 41
Editing Mode 41
Select Node First 42
Select Type of Policy Next 42
Add by Selecting Parameter Type 42
Understand Inheritance for the Node 42
Determine Where to Edit an Inherited Rule 43
Parameter Restrictions 44
Publishing Policies 45
HTTP Request Data Type Parameters 45
Specifying Parameters 47
Protocol 48
Host 48
Path 48
Extension 48
Method 49
Query Parameter 49
Unnamed Query Parameter 49
Path Segment 50
Segment Keys 50
Segment Parameters 50
Segment Ordinals 50
Segment Parameter Ordinals 50
Cookie 51
User Agent 51
Referrer 51
Header 52
Client IP 53

Chapter 5 The Request Type Hierarchy 55
Managing the Hierarchy 56
Hierarchy Inheritance 57
Origin of Rules in a Policy 57

Inheritance Indicators 58
Overriding Inherited Policies. . . . 59
Request Type Hierarchy Example 60
Tasks 60
Actions. . . . 61

Chapter 6 Setting Matching Policies 63
Supported HTTP Request Data Types 64
Supported HTTP Response Data Types 64
Defining Matching Policies 65
Matching Parameters 65
Application Matching and the Request Type Hierarchy 66
Application Matching Precedence Rules 66
Unmatched Requests. . . . 67
Application Matching Example 68
Tasks 68
Actions. . . . 69

Chapter 7 Setting Variation Policies. . . . 71
Content Variation and Caching. . . . 72
Default Caching Behavior 73
When Variation Policies Are Required 73
When Variation Policies Are Optional. . . . 74
Defining Variation Policies 75
Variation Parameters. . . . 75
Value Groups 76
Multiple Matching Parameters 77
Ambiguous Query Parameters 77
Content Variation and the Request Type Hierarchy 78
Content Variation Example 79
Tasks 79
Actions. . . . 79

Chapter 8 Setting Assembly Policies. . . . 81
Assembly Policies 82
Parameter Value Substitution 83
Random Value Substitution 84
Express Loader Management 84
Link Management 85
Express Loader Example 86
Express Connect Management 86
Link Management 87
Express Connect Example 88
Compressing Content 88

Content Assembly on Proxies 89
Rewrite Engine Scripts 90
Advanced Assembly Options 91
Defining Assembly Policies 91
  Parameter Value Substitution Parameters 92
    Substitution Source 92
    Substitution Target 93
    Target URLs 93
Content Assembly and the Request Type Hierarchy 93
Content Assembly Example 94
  Tasks 94
  Actions 95
Using the Debugger 95
  Enabling the Debugger 96
  Accessing the Debugger 96

Chapter 9 Assembling Content with ESI 97
Processing an ESI Response 98
  Caching the Response 98
  Caching Fragments 99
Serving ESI Content from the Cache 99
ESI Templates 99
ESI-Included HTML Fragments 100
ESI Language Definition 101
  ESI Expressions 101
    ESI Literals 102
    Complex Expressions 102
    Operand Associations 102
    Expression Negation 102
  ESI Operators 102
    Binary 103
    Comparison 103
    Logical Operators 104
  ESI Variables 104
    Lists 105
    Dictionaries 105
    Default Values 106
    Cookie Handling 106
  esi:vars 107
  esi:include 107
  esi:inline 108
  esi:try 109
  esi:choose 111
  esi:comment 112

esi:remove . . . 112
<!--esi ... --> . . . 113
Parameter Value Substitution . . . 114
ESI Variables . . . 115
ESI Example . . . 115
Request Type Hierarchy . . . 116
Caching Policies . . . 116
The ESI Template . . . 117
How it Works . . . 118
The HTML Fragment . . . 119

Chapter 10 Setting Proxying Policies . . . 121
Supported HTTP Request Data Types . . . 122
Always Proxy . . . 122
Proxy Rules . . . 123
Defining Proxy Rules . . . 123
Proxy Rule Parameters . . . 123
Proxy Example . . . 124
Tasks . . . 124
Actions . . . 124
Proxy Override Rules . . . 125
Proxy Override Example . . . 125
Tasks . . . 126
Actions . . . 126
Proxying and the Request Type Hierarchy . . . 126

Chapter 11 Cache Control and Content Lifetime . . . 129
Content Lifetime Settings . . . 130
Maximum Age . . . 130
Browser Cache Minimum Age . . . 131
Stand-In Period . . . 131
HTTP Lifetime Heuristic . . . 132
Content Lifetime Mechanisms . . . 132
ESI Surrogate-Control Headers . . . 133
HTTP/1.1 Cache-Control Headers . . . 133
HTTP/1.1 Cache-Response Directives . . . 134
HTTP/1.1 Cache-Request Directives . . . 136
Defining Lifetime Policies . . . 137
Lifetime Policies and the Request Type Hierarchy . . . 138
Lifetime Mechanism Precedence Rules . . . 138
Lifetime Example . . . 139
Tasks . . . 140
Actions . . . 140

Chapter 12 Invalidating Cached Content 143
Overview of Invalidation 144
Understanding Invalidation Rules 144
When Invalidation Rules are in Effect 145
Invalidation Rule Lifetimes 146
Manual Invalidation Rules. 146
Configuring Manual Invalidation 147
Manual Invalidation Example 147
Raising Invalidation Rules with Triggers 147
Defining Invalidation Trigger Policies 148
Sources for Triggers. 149
Targets for Triggers 149
Specifying a Target. 150
Invalidation Triggers and the Request Type Hierarchy. 150
Invalidation Trigger Example 150
Raising Invalidation Rules with ESI 152
Sending ESI Invalidation Requests 152
Authorizing an Invalidation Request 153
Invalidation Response. 153
Creating an ESI Invalidation Request 154
Basic Invalidation Selectors 155
Advanced Invalidation Selectors 156
Invalidation Action 158
ESI Invalidation Example. 159
Selector Attribute Reference 161
URIPREFIX 161
HOST 161
METHOD 162
BODYEXP 162
URIEXP 163
Selector Element Reference 164
COOKIE 164
HEADER 165
OTHER TYPE PV_QUERY_EXP 165
OTHER TYPE PV_UNAME_QUERY_EXP 166
OTHER TYPE PV_COOKIE_EXP 167
OTHER TYPE PV_USER_AGENT_EXP. 168
OTHER TYPE PV_REFERER_EXP 168
OTHER TYPE PV_HEADER_EXP. 169
OTHER TYPE PV_CUSTOM_HEADER_EXP. 170
OTHER TYPE PV_HOST_ALT 171
OTHER TYPE PV_SEGMENT_L_R_EXP. 172
OTHER TYPE PV_SEGMENT_R_L_EXP. 173
OTHER TYPE PV_EXT_EXP 174

Chapter 13 Connection Mapping 175
Managing Connections. 176
Connection Types 176
    Default Connection Type 176
    One-to-One Connection Type 176
    Request-based Connection Type 177
    Managing Request-based Connections 177
    Value Groups 178
Connection Mapping and the Request Type Hierarchy 179
    Multiple Rules-based Parameters 179
Connection Example. 179
    Tasks 180
    Actions. 180

Chapter 14 Setting Responses Cached Policies 181
Managing Responses Cached Policies 182
Content Assurance 182
Authorized Content 182
Response Codes. 183
Responses Cached and the Request Type Hierarchy. 183
Responses Cached Example 184
    Tasks 184
    Actions. 184

Chapter 15 Setting Log Formats 185
Defining Hit Log Formats 186
Pre-defined Hit Log Formats. 186
    Common, Combined, and Extended Formats. 187
Log Headers. 187
Custom Log Formats 188
Hit Log Format Example 189

Appendix A Requirements for Cache Behavior 191
Servicing HTTP Client Requests. 192
Servicing HTTP Client Requests from Cache 192
Caching HTTP Responses 193

Appendix B Rewrite Engine. 195
Customizing Rewrite Scripts. 196
Running Rewrite Scripts 197
    How a Rewrite Script Is Run 197
    Specifying a Rewrite Script to Run 198
Language Processing 198
Language Expressions 199

Procedures 199
Language Literals 200
    Functions 200
Predefined Procedures 205
 Transform 206
Obtaining Information from the Request. 206

Appendix C Response Headers 211
Response Processing 212
    Responses Served from Cache 212
    Responses Served from a Proxy. 212
        Classifying Responses 213
        Rewriting and Assembling Responses 214
    Response Header Processing 215
ESI Surrogate-Control Headers 215
    Supported Control Directives 216
    Unsupported Control Directives 217
    Overriding HTTP Cache-Control Headers 217
    Surrogate Targeting 217
X-PvControl Response Header 218
    Header Format 218
X-PvInfo Response Header 219

Appendix D Regular Expressions 221
Strings 222
Metacharacters 224
Pattern Definitions. 224

Preface

The F5 WebAccelerator is a distributed system that is designed to improve your site's performance while off-loading traffic from your site's origin servers.

This Policy Management Guide describes the concepts, tools, and procedures used for setting caching policies. Caching policies are rules that you create to tailor the WebAccelerator's caching behavior for your site.

This guide provides information on:

- organizing your WebAccelerators into clusters and assigning them to applications
- core concepts and features on creating and administering policies
- Request Type Hierarchy and how it is used to organize and apply your site's policies
- basic caching concepts, including application matching, content variation, assembly, proxying and caching rules, response codes, and content lifetime
- forcing a refresh of cached content, also called content invalidation
- ESI as it relates to content assembly and content invalidation
- performance monitoring
- customizing log formats
- post-processing the response by inserting response headers and using rewrite scripts
- regular expressions and how they are used in the Administration Tool

Related Information

In addition to this guide, there are several other sources of information about the F5 WebAccelerator product:

- the *Getting Started Guide*, which provides information on the initial installation and configuration of the F5 WebAccelerator
- the *Administration Guide*, which provides information on starting and stopping processes, backing up and restoring the WebAccelerator and its database, changing the WebAccelerator configuration file `pvsystem.conf`, and managing log files
- the online help that is part of the Administration Tool user interface, which provides information on each screen that is part of the user interface
- the release notes, which provide information about known issues and workarounds and any information that was too late to be included in the product documentation

Conventions Used in this Book

This section explains the conventions used in this book.

`Monospaced font` – This font is used for examples, text that appears on the screen, command line utility names, and filenames.

`<bracketed text>` or *`italic text`* represents elements in a path or example that are intended to be replaced with information specific to your installation or procedural requirements.

Text of this color indicates a link in PDF or HTML that you can click on to navigate to a related section.

Note: Notes mark important information. Make sure you read this information before continuing with the task.

Technical Support Information

Phone	(+1) 206-272-6888
Fax	(+1) 206-272-6802
Web	http://tech.f5.com
FTP	ftp.f5.com
Email	support@f5.com

Chapter 1

Introduction

- WebAccelerator Components
- Managing Accelerators
- Serving and Caching Pages
- Managing Content
- Customizing WebAccelerator Behavior
- Performance Reports
- ESI Support

The WebAccelerator is a distributed system for accelerating content served by your web and application servers. Using unique algorithms that describe the contents of a web page based on the information found in the HTTP request, the WebAccelerator is capable of efficiently caching, reassembling and serving the content most commonly requested from your website.

Most sites are built on a collection of web servers, application servers, and database servers that we refer to collectively as origin servers. Unlike the F5 WebAccelerator, your origin servers can serve all possible permutations of the content offered by your site. The WebAccelerator only stores and serves page content combinations that were previously requested by visitors to your site.

The first time that a request for a specific page is made, the WebAccelerator proxies the request to your origin server, and attempts to cache the response. The next time that page is requested, the WebAccelerator can serve the response, freeing up your origin servers. By serving the bulk of common requests, the WebAccelerator greatly reduces the load on your origin servers. By working together with your origin servers, it improves performance and distributes the load experienced by your site.

WebAccelerator Components

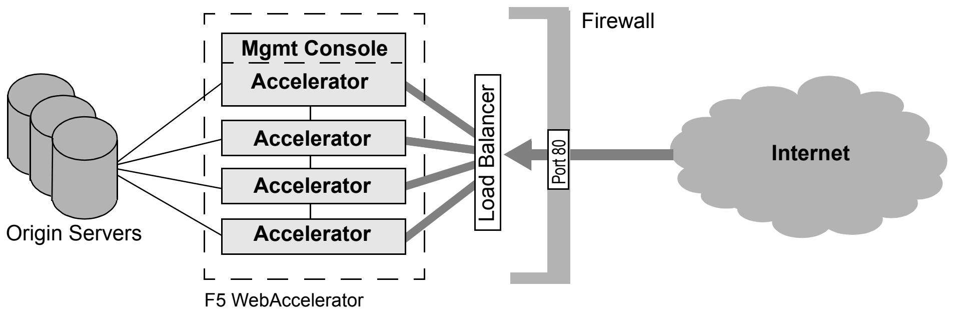
The F5 WebAccelerator consists of two major components: the Management Console and the WebAccelerators. Every WebAccelerator system has one Management Console and one or more WebAccelerators. You cannot have two Management Consoles actively running in a single WebAccelerator system.You can have multiple WebAccelerator systems running in your enterprise, but they are separate systems and do not communicate with each other.

The Management Console controls and manages the WebAccelerator, and provides internal communications. Configuring or logging into the WebAccelerator is done by logging into the Administration Tool (Admin Tool) running on the Management Console. All WebAccelerators that are part of the WebAccelerator system receive their configuration from the Management Console.

The WebAccelerators act as the servers: analyzing incoming requests, then caching and serving responses or redirecting requests to the origin servers as needed.

The WebAccelerator can be on one machine or distributed across many machines. If the WebAccelerator is installed on a single machine, there is one WebAccelerator and one Management Console on that machine. This is a very simple model with no redundancy and no fault tolerance. The more typical installation is a WebAccelerator with multiple WebAccelerators. If you have multiple WebAccelerators, one WebAccelerator is always on the same machine as the Management Console. The other WebAccelerators are each on their own machine:

Figure 1 Basic Multiple WebAccelerator Configuration



The F5 WebAccelerator Remote is a special type of WebAccelerator that can be located at a remote location to improve performance for users at that remote site. If the WebAccelerator Remote cannot service a request from cache, it proxies the request to the WebAccelerator accelerating the requested application, instead of proxying to the origin server. It is specially designed for communication with other WebAccelerators, but you configure and manage it through the central Management Console just like any other WebAccelerator.

For information on installation of the WebAccelerator software and new features, see the *Getting Started Guide*.

Managing Accelerators

WebAccelerators are managed as clusters. After installing an WebAccelerator, you log into the Admin Tool running on the Management Console and assign the WebAccelerator to a cluster. A cluster can have one or more WebAccelerators. You configure the cluster, rather than individual WebAccelerators, simplifying the configuration and management of your WebAccelerators.

After you have created your clusters, you assign them to your applications. Each application is assigned one cluster. When you assign your WebAccelerators to clusters, be sure that all WebAccelerators you want assigned to a particular application are in the same cluster. You can assign a single cluster to several applications.

You determine how many clusters you need and which WebAccelerators should be in which cluster. The grouping is arbitrary. You might create clusters organized by location or application. For more information, see "WebAccelerator Clusters" on page 20.

Serving and Caching Pages

The first time the WebAccelerator receives a request for a particular piece of content, it proxies to the origin servers for that content. When the WebAccelerator receives the page, before sending the response to the client, the WebAccelerator:

- transforms a copy of the page into a compiled internal representation

  WebAccelerators use these compiled responses to recreate a page in response to an HTTP request. Compiled responses consist of:

  – an internal representation of the page as it was received from the origin site

  – instructions which describe how to recreate the page from the internal representation, including information needed to update the page with any random or rotating content. See Chapter 8, Setting Assembly Policies.

- assigns a Unique Content Identifier (UCI) to the compiled response, based on elements present in the request

  Certain elements in a request, such as the URI, query parameters, and so on, are what generated this particular response from the origin servers. Other requests containing the same elements should be served the same response. Using these elements as part of the identifier makes it easy to match future requests to the correct content in cache. This UCI is used for both the request and the compiled response that is created to service the request.

- caches the compiled response under the UCI, so when a future request that contains the same elements and generates the same UCI is received, this cached response is found and used to respond to the request

Not all elements in a request indicate a unique response. For example, two requests with the same URI, same method, and same query parameters but different cookies might still generate an identical response from your origin servers. By default, the WebAccelerator assumes that certain elements cause the content to vary. It also assumes that certain other elements do not cause the content to vary. You can change these assumptions with variation policies. See Chapter 7, Setting Variation Policies.

Managing Content

After the WebAccelerator is installed and the initial configuration completed, it can start handling requests and managing content. The initial configuration involves:

- creating WebAccelerator clusters
- creating application profiles, including the host map
- selecting policy sets

This provides everything the WebAccelerator needs to operate.

The host map created during initial configuration is used by the WebAccelerator to redirect or proxy requests to your origin web servers if it cannot service the request from its own cache. The host map is explained in "Host Maps" on page 21.

The policy set is loosely divided into matching policies and caching policies. The matching policies are used to match requests to certain sets of caching policies, based on characteristics of the request. Those caching policies are then applied to the handling of the request and the response.

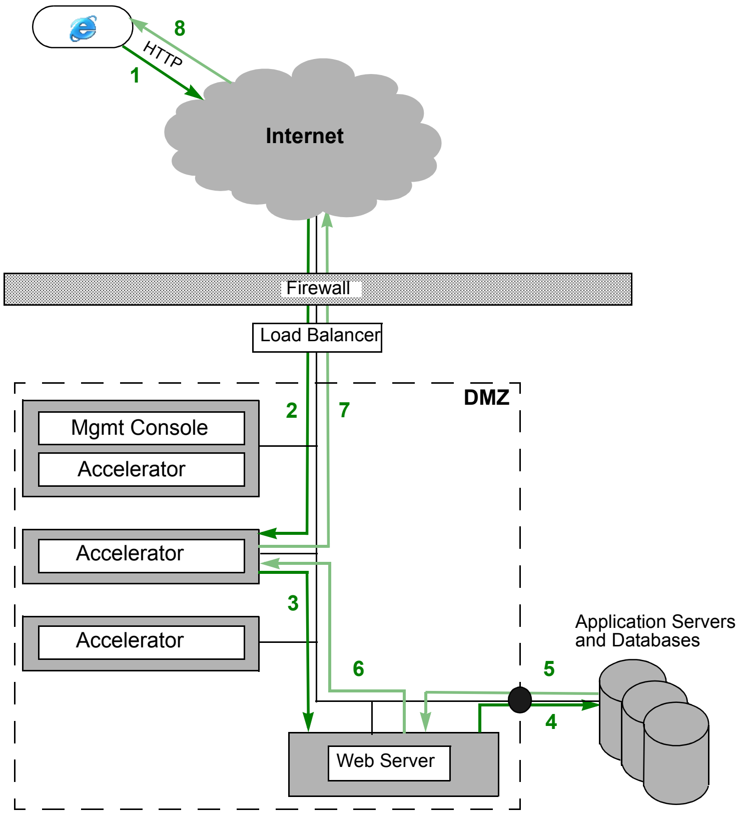
Anything in this initial configuration can be customized or replaced to optimize request/response handling for your site. The initial configuration and any modifications to this configuration are performed using the Admin Tool in the Management Console component of the WebAccelerator. You can access the Admin Tool through your web browser by entering its URL. The Management Console distributes the configuration to all the WebAccelerators that are part of that WebAccelerator system.

The request and response flow as an WebAccelerator handles a request follows this general pattern, illustrated in Figure 2 on page 5:

1. Pages are requested from your site as usual by browsers and other web clients. From their perspective, they are connecting directly to your site – they have no knowledge of the WebAccelerator.
2. The request is directed to one of the WebAccelerators, which examines the request to determine if it meets all the requirements needed to attempt to service the request from a compiled response in its cache. If it does not, the WebAccelerator redirects the request to your origin servers and the origin servers respond to the request.

   For information on the requirements that an HTTP request must meet, see "Servicing HTTP Client Requests" on page 192.

Figure 2 Request/Response Flow



3. If the request can be serviced from a compiled response, the WebAccelerator creates a UCI for the request, based on elements it sees in the request, and looks in its cache to see if it has a compiled response stored under that same UCI:
   - if the page is being requested for the first time (no matching compiled response in cache), the WebAccelerator immediately proxies the request to the appropriate web server, based on the host map
   - if a page with the same UCI is already stored as a compiled response in the WebAccelerator's cache, the WebAccelerator skips to Step 7
4. The web server either responds or queries the application servers or databases that need to provide input to the page. These application servers or databases might be in a more secure location, outside the DMZ.
5. The application servers or databases provide the input back to the web server.
6. The web server replies to the WebAccelerator with the requested material and the WebAccelerator compiles the page into a compiled response. If the response meets all the appropriate requirements ("Caching HTTP Responses" on page 193), the WebAccelerator stores the compiled response in its cache under the appropriate UCI.
7. The WebAccelerator uses the compiled response to create the page and respond to the request.
8. The response is directed to the originator of the request.

Customizing WebAccelerator Behavior

You customize how the WebAccelerator handles your site's HTTP requests by changing or adding policies. You can change which policy set is used to handle requests for an application. You can also create a custom policy set, containing:

- Application Matching Policies, which determine how incoming requests are matched to caching policies

  Changing the matching policies changes which caching policies are applied to a request and response.
- Application Caching Policies, which determine how the WebAccelerator handles an HTTP request

  There must be a set of caching policies to correspond with each matching policy. In other words, if a matching policy matches a request or a response, there must be a set of caching policies to apply to that request or response.

You can also control some aspects of the WebAccelerator's behavior using Edge-Side Include (ESI) tags. For an overview of the WebAccelerator's support for ESI, see "ESI Support" on page 11.

The first step in customizing WebAccelerator behavior is to analyze the sorts of HTTP requests your site receives, and determine how you want them to be handled. Which requests have cache-able responses? Which requests have responses which should never be cached? Which requests are for static documents that can remain in cache for several days, and which are for documents that should be refreshed hourly?

Basically, you are deciding how many different sets of application caching policies you need. Once you decide on the different ways the WebAccelerator should handle requests to your site and responses to those requests, you need to determine how the WebAccelerator can correctly differentiate between requests, so that it can apply the appropriate set of caching policies. For example, maybe the path found on requests for static documents is different than the path for dynamic documents. Or maybe the paths are similar, but the static documents are PDFs and the dynamic documents are Word documents or Excel spreadsheets. These differences help you create matching policies that can be applied to an incoming HTTP request and correctly match requests that should be handled by particular set of caching policies.

Depending on how your site works, information in requests can sometimes imply one type of response (perhaps the file extension is `.jsp`), when the actual response is a bit different (perhaps a simple document). Matching policies are applied twice, once to the request and a second time to the response. This means a request and a response can be matched to different sets of caching policies, but ensures the response is matched to the policies best suited to it.

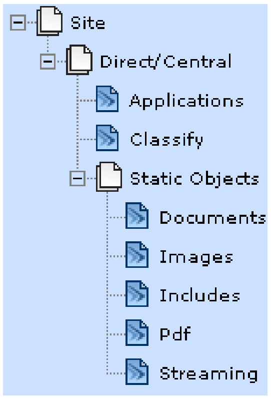
Application Matching Policies

Application matching is the process of mapping HTTP requests and responses to the relevant caching policies. Matching policies are used to group requests and responses. Sets of caching policies are associated with each group. Once requests and responses are grouped, the relevant set of caching policies can be applied.

Information in the request or response is analyzed and compared to each matching policy. Each matching policy is made up of one or more rules, enabling you to finely differentiate between requests and responses. Rules can specify things like the path, file extension, cookie, or query parameter the matching policy looks for on a request.

The matching policies are organized in a tree structure called the Request Type Hierarchy. Each matching policy is a node on that tree. When a request or response is matched to a matching policy, it is assigned to that node on the tree. By organizing the policies in this way, one policy can inherit rules from its parent or ancestor policies, which makes it easy for you to define a rule that should be included in several policies. The tree is also used to associate the caching policies with the matching policies. You create caching policies for each node, and they are the policies applied to requests or responses assigned to the node.

Matching is only performed against leaf nodes. Requests and responses can not be matched to a matching policy on a parent node. The primary purpose of parent nodes is inheritance. For example, the figure shows a sample Request Type Hierarchy as it might appear in the Admin Tool:



The matching policy for Documents inherits rules from Static Objects, Direct/Central, and Site. Requests can only be matched to the leaf nodes, which are Applications, Classify, Documents, Images, Includes, Streaming, and pdf. If a request matches the rules in Static Objects, but not any of the additional rules found in the leaf nodes under Static Objects, the request is considered unmatched (see "Unmatched Requests" on page 67).

When you create a matching policy, you start by:

1. Creating a leaf node for it on the Request Type Hierarchy tree. If you want the policy to inherit rules from an existing parent policy, you can add it as a leaf node under that parent.
2. Defining rules for the matching policy that identify the requests which you want to handle in a particular way.
3. Creating a set of caching policies designed to handle the requests in the way you want, associating the caching policies with this node.

During application matching, if a request matches your new matching policy, it is assigned to this node, and the caching policies associated with this node are applied to the request.

Matching occurs twice during processing. Request matching matches requests to a node. If the request is going to be proxied, some of the caching policies related to proxying are applied. When the response from the origin servers is received, response matching matches the response to a node. It might match to the same node as the request, in which case the remainder of the caching policies for that node are applied. In some cases, the response might match to a different node, and a subset of the caching policies from the new node are applied.

For example, a simple site might expect only two types of requests, for GIF images and for a CGI-based application. You create your Request Type Hierarchy to have two nodes, Images and Applications, with the relevant matching policy for each. One policy identifies appropriate requests by using a rule that specifies a match if the file extension is `.gif`. This policy matches requests to the Images node. Another policy identifies requests for the CGI-based application using different rules:

- one rule might be based on the path that appears in the requesting URL
- another rule might be based on the request and response not matching an image content type

This policy matches requests to the Applications node. However, if a request matches to the Applications node because its path is for the CGI-based application, but the response that comes back is a GIF image, the response is now matched to the Images node.

For more information on:

- the Admin Tool, see Chapter 2, Using the Administration Tool
- the HTTP request data types, see “HTTP Request Data Type Parameters” on page 45
- the Request Type Hierarchy, see Chapter 5, The Request Type Hierarchy
- application matching, see Chapter 6, Setting Matching Policies

Application Caching Policies

When the WebAccelerator receives an HTTP request, it determines:

- whether the request is for cache-able content
- whether the request is for content that has already been cached
- whether the content it has cached is fresh enough that it can be used to service the request
- how to compile responses from origin server content
- how to reassemble compiled responses into the appropriate web page needed to service the request

The actual behavior for each of these items is set by the rules contained in the application caching policies. Some of these policies are relevant to how a request is handled, and some are more relevant to the handling of the response.

You can set caching policies that:

- indicate whether specific elements seen in the HTTP request affect the content contained in the web page and should be used in the UCI to identify cached content. This policy is applied to requests. See Chapter 7, Setting Variation Policies.
- identify the portions of the web page that the WebAccelerator should vary for you when it assembles the content from the compiled response. This policy is applied to responses. See Chapter 8, Setting Assembly Policies.
- force the WebAccelerator to always forward the request to your origin servers. Parts of this policy are more relevant to requests, and parts are more relevant to the handling of the response. See Chapter 10, Setting Proxying Policies.
- specifies which content the WebAccelerator should cache based on the HTTP response code that arrived with the content. This policy is applied to responses. See Chapter 14, Setting Responses Cached Policies.
- specifies how long content remains in the cache without a refresh by specifying the minimum and maximum age the WebAccelerator allows for cached content. You can also specify the mechanisms that the WebAccelerator uses to arrive at these values. This policy is applied to responses. See Chapter 11, Cache Control and Content Lifetime.
- specifies the conditions under which the WebAccelerator should automatically invalidate cached content.

  Various cache invalidation mechanisms allow you to refresh cached content before it expires. You can manually invalidate content or set invalidation triggers. If content expires, the next time that content is needed to service a request, the WebAccelerator proxies to the appropriate origin server for fresh content. This policy is applied to requests. See Chapter 12, Invalidating Cached Content.

Each caching policy you create is associated with a node on the Request Type Hierarchy. The caching policy is applied to the request or response that matches to the node based on the matching policy for the node. Just like the matching policies, the caching policies on a leaf node can inherit policies from its ancestors.

Performance Reports

The WebAccelerator includes the Performance Reports feature that makes it simple to monitor the server’s performance. It enables you to evaluate cache efficiency so you can determine the effectiveness of your policies. The interface is part of the Admin Tool and displays graphical and table views of the cache hit, miss, and proxy data, in addition to other useful information and statistics for the WebAccelerator system.

Use the Performance Reports tool to view statistics and generate charts and reports on the WebAccelerator's performance and traffic load. You can change the type of report by choosing a new report from the list on the left. For example, you can change from Hits by Application to Load Time by Content Size.

The Edit Filter function allows you to change the filter criteria, including the time period covered by the report. The filter criteria stay in effect until you change them, even if you switch to a different type of report.

The first time you use Peformance Reports, you should edit the filter to ensure the reports make sense for your applications. It is often helpful to use the Transaction Type selection box to choose an application or nodes for an application as part of the filter. Then all the data you see is for that application.

You can change the time period by selecting a time period from the drop-down list. If you select `Custom...` from the drop-down list, a calendar lets you select the dates for the time period you want. Any invalid dates are greyed out in the calendar. Invalid dates are dates for which there is no historic data, such as any time in the future or dates in the past for which data is no longer saved. Because the WebAccelerator collects so much system data, it cannot store historic data for more than a few weeks.

You can change which data is displayed by selecting certain filter criteria:

- Cluster, where you choose which clusters to show data for
- Content Type, where you choose which responses to show data for, based on how they were classified by type of file. Content type represents the object types defined in the global fragment configuration file. For more information, see "Classifying Responses" on page 213.
- Content Size, where you choose which responses to show data for, based on the size of the response
- Transaction Type, where you choose which responses to show data for, based on the node they matched against in the Request Type Hierarchy during response matching

ESI Support

Edge-Side Includes (ESI) is an open specification for supporting web surrogates such as the WebAccelerator. It comes in two parts:

- ESI as used for page assembly is an XML-based language that identifies page fragments and other resources to be included during page assembly. The WebAccelerator supports the entire 1.0 ESI specification. See Chapter 9, Assembling Content with ESI.

A secondary part of using ESI for page assembly is the ability to use ESI response headers to identify cache lifetime or no-cache conditions. See "ESI Surrogate-Control Headers" on page 215.

- ESI as used for cache invalidation defines an XML Document Type Definition (DTD) that is used to define invalidation objects. You send one or more invalidation objects to the WebAccelerator using an invalidation request. Invalidation requests are sent to the WebAccelerator using the HTTP/1.1 protocol.

  The WebAccelerator raises one invalidation rule for each invalidation object it sees on an ESI invalidation request.

  The WebAccelerator supports the entire 1.0 version of the ESI Cache Invalidation specification. For more information, see "Raising Invalidation Rules with ESI" on page 152. For information on invalidation rules in general, see Chapter 12, Invalidating Cached Content.

Chapter 2

Using the Administration Tool

Accessing the Admin Tool ◂
Navigating the User Interface ◂
Starting at the Home Page ◂

When you install the Management Console component of the WebAccelerator, it includes the HTTP-based user interface called the Administration Tool (Admin Tool). During installation, you specify the hostname to be used to access the Admin Tool from a web browser. After installation is complete, you enter that hostname as the URL in your web browser and you can begin configuration of your WebAccelerator system.

Much of the WebAccelerator's behavior is managed using the Admin Tool. Extensive online help is provided for each screen in the tool, including links to this book. The Admin Tool provides a simple and intuitive way to manage your WebAccelerator. Almost all configuration is performed through its interface. There are only a few configuration tasks not performed using the Admin Tool:

- tailoring the configuration file, `pvsystem.conf`, found on each WebAccelerator machine (see the *Administration Guide*)
- creating rewrite scripts to do post-processing of responses received from your origin servers (see Appendix B, Rewrite Engine)

The Admin Tool also contains a performance monitor called Performance Reports which you can use to monitor and assess how your policies are working.

Accessing the Admin Tool

To access the Admin Tool, enter the URL of the WebAccelerator Management Console in your web browser. For example:
`https://mynwa.mydomain.com:8443/`

where you enter the real subdomain and domain of your WebAccelerator Management Console instead of `mynwa` and `mydomain`. If you have an unusual installation of the WebAccelerator where SSL is not enabled, use `http` instead of `https` and use port `8080`. To make the Admin Tool easy to access, add the URL to your Favorites list in your browser.

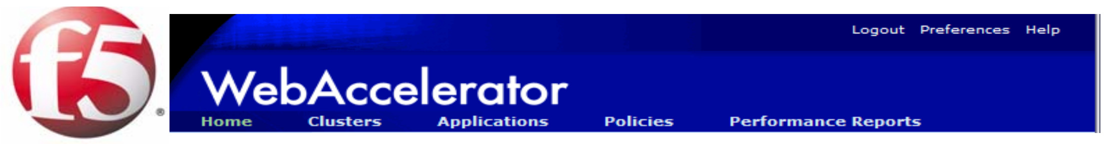
Using your browser to navigate to the Admin Tool takes you to the login screen, where you see a dialog box asking for a username and password. The username is `administrator`. The initial password was set during installation by whoever installed the WebAccelerator, and can be changed under Preferences after logging in.

Your Admin Tool session is automatically logged out after a period of inactivity, and any unsaved changes are lost. The default for this session timeout period is 30 minutes. To change the session timeout period, you edit the `SESSION_EXPIRES_TIME` setting in the `pvsystem.conf` file. See the *Administration Guide* for information.

Navigating the User Interface

There are two main menu bars in the Admin Tool user interface:

Figure 3 Example of Menu Bars



- the basic menu bar at the top right of the screen, which provides these links:
  - Logout, logs you out of your session and takes you back to the login page

  - Preferences, opens a dialog box where you can set your preferences for time zone and change your password
  - Help, brings up online help for the page you are on, together with the table of contents for all the help so you can navigate to any help topic. The table of contents also contains a link to the books online.
- the function menu bar at the left of the screen under the F5 logo, which has links to these functional areas:
  - Home, which is your home page, described in "Starting at the Home Page" on page 18
  - Clusters, where you add WebAccelerators to the WebAccelerator, assign them to clusters, and check their status, described in "WebAccelerator Clusters" on page 20
  - Applications, where you define the applications to the WebAccelerator that you want it to accelerate, including assigning clusters and policy sets to each application, described in "Application Profiles" on page 20
  - Policies, where you edit and customize policy sets and link to the Policy Editor to create new policies, described in Chapter 4, Managing Policies
  - Performance Reports, where you can monitor various aspects of your WebAccelerator system, including requests per minute, cache hits, cache misses, and errors

There is also a bottom bar area on each screen. This area is expandable and collapsible. If you click on the bar, it expands to show more information about the screen. Clicking again on the bar collapses the text area back to a bar.

For most screens that are functional subsets, there is a "Back to" link which takes you back to the main screen for the function. For example, the screen in Figure 3 on page 14 is the summary screen for Destination Hosts, which is a subset of Applications. You can see in the figure that there is a "Back to Applications" link just above the table.

Common Tasks

Many of the tasks you do in the Admin Tool user interface are common across many functions, so they are explained here instead of repeated in descriptions of the functional areas.

Save

The Save button saves any changes you have made, regardless of whether you edited an existing object or created a new one. In general, you cannot revert to the previous state after you click Save.

The OK button is similar to Save, saving any changes you made in a dialog box and closing the dialog box.

Cancel

The Cancel button causes any changes you made since the last save to be ignored and returns you back up one screen level or closes the dialog box.

Refresh

Most Performance Reports are static, reflecting the data when you first created or opened them. Click the Refresh button to resample the data and get a current view. On some screens, there is a Refresh button you can use to update the screen after saving any changes you made to data on the screen.

Add or Add More

The Add and Add More buttons let you begin to add a new parameter or option by opening a screen or text area on a screen where you can input information for the parameter or option you are adding. Add is not complete until you have specified all the required values and clicked the Save button.

If you need to select the type of parameter to add, you must select that first. The Add button is greyed out until you make the selection. After you select the parameter type, you can click Add to go to a screen where you can fill in values for that parameter.

Edit

The Edit link appears in many tables. Clicking on this link takes you to a screen where you can edit fields for the item. Click the Save or Cancel button to exit from the editing screen, depending on whether you want to save your changes or not.

View

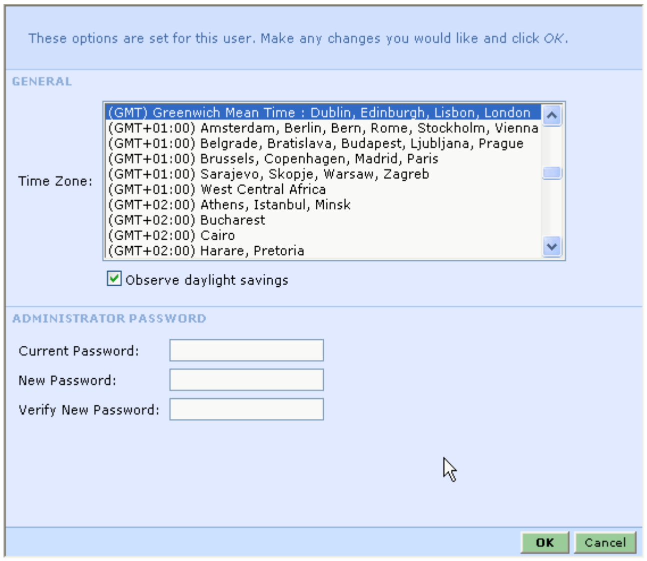
The View link appears in many tables. Clicking on this link takes you to a viewing screen where you can see information for this item in more detail. You cannot change the information for an item when you are on a viewing screen.

Remove or Delete

The Remove and Delete links appear in several tables. Clicking on this link deletes the item completely.

Preferences

In this dialog box, you can set your user preferences:



These options are set for this user. Make any changes you would like and click *OK*.

GENERAL

Time Zone:
(GMT) Greenwich Mean Time : Dublin, Edinburgh, Lisbon, London
(GMT+01:00) Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna
(GMT+01:00) Belgrade, Bratislava, Budapest, Ljubljana, Prague
(GMT+01:00) Brussels, Copenhagen, Madrid, Paris
(GMT+01:00) Sarajevo, Skopje, Warsaw, Zagreb
(GMT+01:00) West Central Africa
(GMT+02:00) Athens, Istanbul, Minsk
(GMT+02:00) Bucharest
(GMT+02:00) Cairo
(GMT+02:00) Harare, Pretoria

☑ Observe daylight savings

ADMINISTRATOR PASSWORD

Current Password:
New Password:
Verify New Password:

You can change your preferences at any time. Click the OK button to save your changes and exit, or click the Cancel button to exit without saving the changes.

Time Zone

Choose your preferred time zone for your Admin Tool sessions, and check the box if you want it to reflect daylight savings time.

Administrator Password

Use this section to change or reset your password. Enter your current password first. Next, enter the new password in both boxes, to verify it. The initial password for Administrator is an option set during installation.

Help

To open Help, click on Help in the basic menu bar. The Help window appears on top of the Admin Tool window you are viewing. If you click on a link in the Admin Tool when the Help window is open, the Help window stays open but is sent to the background and might be covered by the Admin Tool window.

The help is partially contextual. When you open Help, you are shown the main topic for the Admin Tool screen you were on when you clicked Help. If you leave the Help window open and go to a different Admin Tool screen, the help topic shown does not change. To see the topic for the new screen you are on, close the Help window and reopen it after you switch Admin Tool screens.

To navigate to different topics in Help, you can:

- follow links in the help topics

  Use the Back button in the top of the Help window after following a series of links to navigate back to the original topic. If you click on a link to a book, rather than a help topic, a new window is opened for the book information and you simply close that window to go back to your help topic.
- click on topics in the help table of contents
- follow links in the help index, which you can access by clicking on its link in the help table of contents

Some links in the help take you to more detailed information in the product documentation. These links open a new window, so that you can view the product documentation and help information simultaneously. These links are shown in italics.

Starting at the Home Page

The home page provides, immediately upon login, a quick look at WebAccelerator status. From the home page you can navigate to any other area of the Admin Tool. If you simply want to access the product documentation online, go to Help and select Product Documentation from the table of contents. If you want to report a problem, go to Help and select Swan Labs Support from the table of contents. The email link and phone number of the Swan Labs Customer Confidence group are displayed.

Chapter 3

Managing Applications

WebAccelerator Clusters
Application Profiles
Host Maps
Request Handling
Creating a Cluster
Creating an Application Profile
Defining Destination Hosts
Managing Express Connect

When you install the WebAccelerator, it should become the destination of HTTP requests instead of your origin servers. Basically, the WebAccelerators become the recipients of your HTTP traffic. You set up WebAccelerator clusters and assign them to your applications. You reconfigure DNS so the URLs your users specify actually point to the Accelerator clusters handling requests for those applications.

In addition to reconfiguring DNS, you map HTTP domain names to the actual location where your web servers reside. This is done in your application profile. This mapping must include all domain names that you expect your Accelerators to see on incoming HTTP requests. The application profile also identifies which policy set the cluster should use to handle requests.

Once you have defined your WebAccelerator clusters and application profiles, the WebAccelerator has all the information it needs to start handling requests for your applications.

WebAccelerator Clusters

Every Accelerator must be assigned to a cluster. Clusters are arbitrary groupings of Accelerators. You can have as many clusters as you like, and a cluster can have one or more Accelerators assigned to it. An Accelerator can only be assigned to one cluster.

Set up your clusters to organize your system in a way that makes the most sense. You might group WebAccelerators by physical location, or by application that they would accelerate. Keep in mind that an application can only have one cluster assigned to handle its requests. All the Accelerators that you want assigned to an application must be in the same cluster. Any Accelerator not in an application's cluster that receives a request for the application forwards the request to the cluster. Although an application can only have one cluster, a cluster can support multiple applications.

For example, if you have eight Accelerators in your Bethesda data center, you could put them all in a cluster called `Bethesda` even though they support several different applications. However, this would not work if you have an Accelerator in your Washington D.C. office that also should handle requests for one of those applications. In this case, define a cluster for that application and assign the Washington D.C. Accelerator and an appropriate subset of the Bethesda Accelerators to that cluster.

Application Profiles

An application profile provides all the information the WebAccelerator needs to appropriately handle requests for a particular application. For each application you want the WebAccelerator to accelerate, you provide:

- a unique name that identifies the application in the Admin Tool user interface
- a description
- the name of the WebAccelerator cluster accelerating this application, that is, the cluster to which requests for the application are directed
- the name of the policy set that the assigned cluster should use when handling requests for this application
- the name of the policy set that any other cluster should use when handling requests for this application. Generally , these clusters forward the requests to the assigned cluster.
- the host map, which tells the cluster where to proxy for content to service the request

Host Maps

You map domain names by creating a host map. You identify domains as you expect them to appear on the HTTP HOST request header. These domains are called Requested Hosts. You then identify the hostname of your origin servers that would have serviced this request. These are called Destination Hosts.

Each application has its own host map with a unique set of Requested Hosts. When a request is received by an WebAccelerator, the WebAccelerator compares the value of the HOST request header against all the Requested Hosts in all the host maps for this WebAccelerator system. When it finds the best match, the WebAccelerator determines in which host map the match is, and to which application the host map belongs. This is how the WebAccelerator determines which application the request is for, and is why a Requested Host must be unique to a single application. See "Request Handling" on page 23.

When you set up your host mapping and specify the Destination Hosts, you are telling the WebAccelerator where to go for content. It looks to the Destination Hosts for the initial content needed to respond to a request, for the content to refresh its cache, and as the location to proxy a request when it cannot service the request from its cache.

When you add your origin servers to the host map as Destination Hosts, you create a definition for each server. This definition specifies the Host Type of your origin server, which can be an origin web server or proxy server. You can also set various connection values and timeouts in the definition. These properties allow the WebAccelerator to work as efficiently as possible with your origin servers.

Unmapped Hosts

When the WebAccelerator receives a request for a domain not found in the Requested Hosts, it is called an unmapped host request. By default, the WebAccelerator responds to the requesting client with an HTTP 403 response code.

Occasionally you might notice a host map of type Unmapped when creating or testing policies. This occurs when someone creates a matching policy based on a hostname that is not identified in a host map. You can add the hostname to the host map for the applications using this policy set.

You can also specify that the cluster process unmapped host requests instead of responding with an error. Check the box in the Unmapped Hosts Settings area on the Edit or Create Cluster screens. By default, this box is unchecked.

When this option is enabled, the cluster processes unmapped requests in this manner:

- the cluster applies the appropriate policies belonging to the policy set you selected for unmapped host requests

- the cluster either does a DNS lookup and sends on the request or forward proxies the request to the proxy server you specify

Security Implications

Security is a concern if you allow the WebAccelerator to forward any unmapped host requests. If you check the Unmapped Hosts box and do not specify a proxy server, you enable the WebAccelerator to be used as a relay. Unless the WebAccelerator and your origin servers are private and heavily protected, we do not recommend this option. If you check the Unmapped Hosts box and also specify a proxy server to forward the requests to, and the proxy server is set up to deal with unwanted or unsanctioned requests, this might be an acceptable method to handle requests for unmapped hosts.

In general, Swan Labs recommends mapping all potential hosts found in legitimate requests for your applications in your host maps, and using the default of responding to requests for unmapped hosts with a 403 response code. You can use wildcards in your host maps to easily map a set of subdomains to a single origin server.

Valid Hostnames

The hostname of a domain is broken into subdomains, separated by periods, as in `www.123corp.com` or `catalog.sales.bigretailsite.com`. When you specify a hostname for a Destination host, you must specify the complete hostname, because the WebAccelerator must know exactly where to proxy for content. When you specify the hostname for the Requested Host, you can use an asterisk (*) as a wildcard for the first part of the domain. The asterisk must be the first character in the name and must be followed by a period. It can represent one or more subdomains. This allows you to map several subdomains to one origin web server in one step, a time-saver if your site has many subdomains.

Valid examples of a wildcard in the Requested Host name are:

- `*.sales.123corp.com`, which maps these to the same Destination Host:
  ```
  direct.sales.123corp.com
  marketing.sales.123corp.com
  marcom.marketing.sales.123corp.com
  ```
- `*.123corp.com`, which maps these to the same Destination Host:
  ```
  www.123corp.com
  engineering.123corp.com
  direct.sales.123corp.com
  marketing.sales.123corp.com
  marcom.marketing.sales.123corp.com
  ```
- `*.com`, which maps all incoming requests that end in `.com` to one Destination Host
- `*`, which maps all incoming requests to one Destination Host

These mappings have one exception: if several of the Requested Host hostnames could match a request, the most specific match is chosen. For example, if you define these Requested Host hostnames:

```
a.com
www.a.com
*.b.a.com
*.a.com
```

When requests are received that contain these URLs, they are mapped to the Requested Hosts as follows:

- a request to www.a.com is mapped to `www.a.com`, not to `*.a.com`
- a request to a.com maps to `a.com`
- requests to c.a.com and b.a.com both map to `*.a.com`
- a request to c.b.a.com maps to `*.b.a.com`

This allows you to define several specific Requested Host hostnames, and a much more general Requested Host hostname to map all other requests. For example, requests directed to your internal HR and payroll applications should go to their own Destination Host, and all other requests go to a different Destination Host. You define your host map as:

Requested Hosts	**Destination Hosts**
`*.hr.inside.anycompany.com`	`hr.anycompany.com`
`*.payroll.inside.anycompany.com`	`payroll.anycompany.com`
`*.anycompany.com`	`primary.anycompany.com`

Request Handling

Incoming requests to an WebAccelerator are first matched to a Requested Host. This match is the mechanism for mapping the request to the application. Each application has its own host map, with a unique set of Requested Hosts. When the WebAccelerator matches the request's HOST value to a Requested Host, the WebAccelerator now knows the host map in which the matching Requested Host was found, and to which application the host map belongs. Once the application is identified:

- the application name is added to the UCI of the request, in addition to elements found on the request
- the information in the application's profile is used to handle the request, such as which set of policies to apply

The process an WebAccelerator uses to handle a request is described in Figure 4.

Figure 4 Request Flow

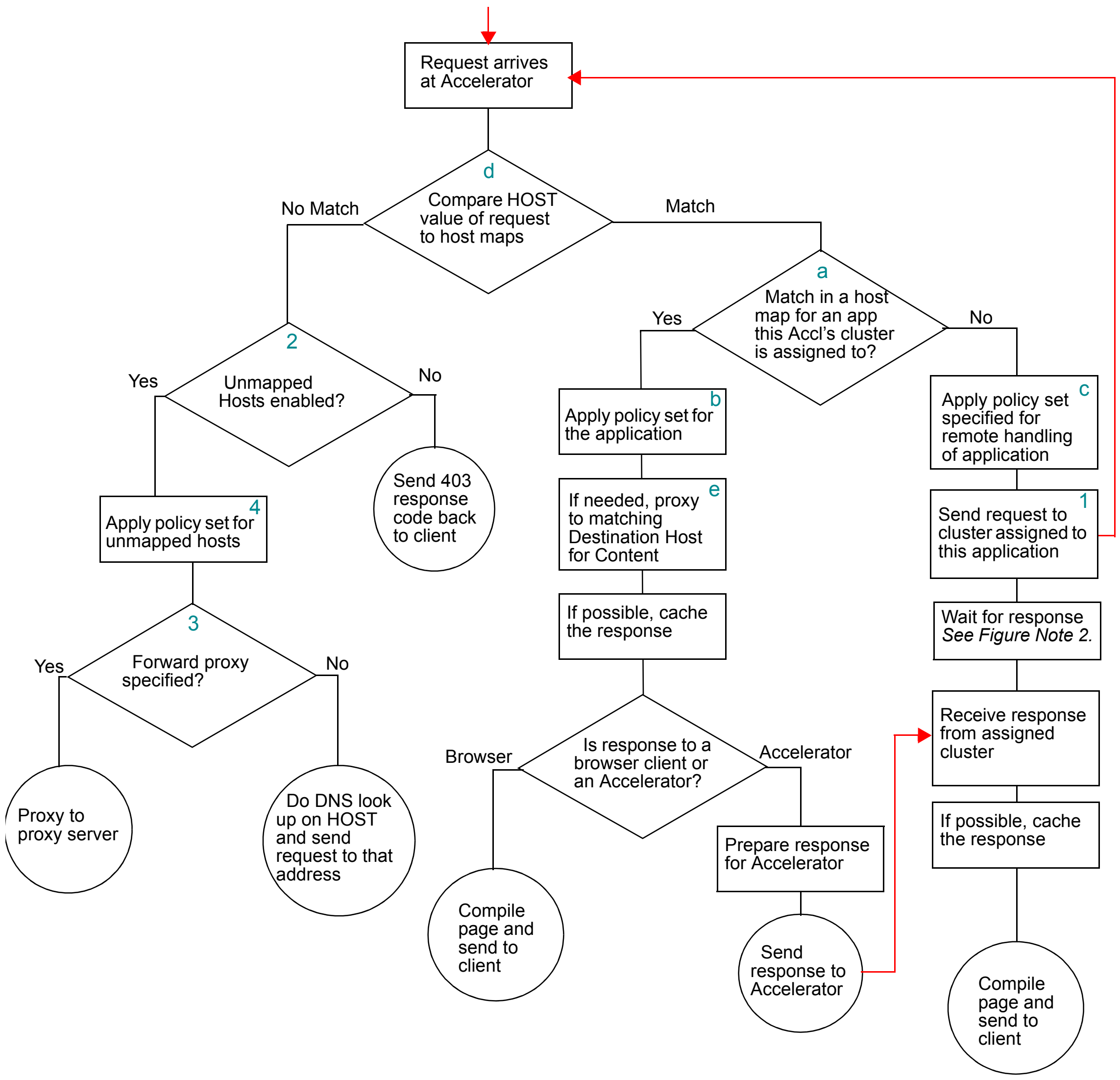

Request arrives at Accelerator
d
Compare HOST value of request to host maps
No Match
Match
a
Match in a host map for an app this Accl's cluster is assigned to?
Yes
No
2
Unmapped Hosts enabled?
Yes
No
b
Apply policy set for the application
c
Apply policy set specified for remote handling of application
Send 403 response code back to client
4
Apply policy set for unmapped hosts
e
If needed, proxy to matching Destination Host for Content
1
Send request to cluster assigned to this application
If possible, cache the response
Wait for response See Figure Note 2.
3
Forward proxy specified?
Yes
No
Receive response from assigned cluster
Is response to a browser client or an Accelerator?
Browser
Accelerator
Proxy to proxy server
Do DNS look up on HOST and send request to that address
Prepare response for Accelerator
If possible, cache the response
Compile page and send to client
Send response to Accelerator
Compile page and send to client

Figure Notes:

1. The callouts in the figure are referred to later in the step-by-step instructions that describe how you specify options for clusters and applications. Callout numbers

reflect information you specified when defining a cluster. Callout letters reflect information you specified when defining an application profile. See "Creating a Cluster" on page 25 and "Creating an Application Profile" on page 28.

2. Some WebAccelerator systems are widely distributed, with WebAccelerators at different locations accelerating different applications, and WebAccelerators located at remote locations mainly to support those remote users. Remote WebAccelerators can also accelerate applications located at those locations. Depending on how you configure DNS and routing, requests from users at remote locations can first be routed to their local WebAccelerator. Frequently, this is not the WebAccelerator accelerating the application for which the request is intended. That is, the WebAccelerator does not belong to the cluster assigned to the application. In this case, the WebAccelerator applies the remote policy set and sends the request on to the appropriate cluster, which handles the request like any other. The process of passing along the request is illustrated in the figure by the red lines.

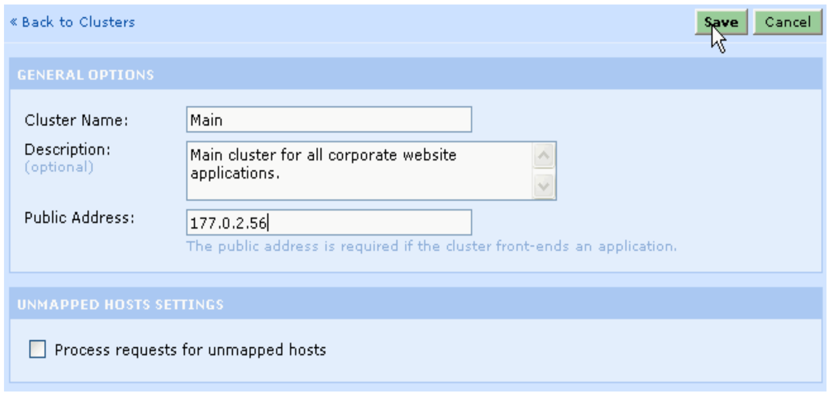
Creating a Cluster

The first step in configuring the WebAccelerator is to define your WebAccelerator clusters. Before you can assign an WebAccelerator to a cluster, it must be installed and running, and the Management Console must be able to detect it on the network. When you log into the Admin Tool, the WebAccelerator should appear on the Clusters screen in standby mode. It does not become active until it is added to a cluster.

Step 1 Access Clusters screen

1. Log into the Admin Tool.
2. Click on the Clusters menu item.
3. Click on the Create New Cluster link.

Step 2 General Options for clusters



Cluster Name: Main

Description: (optional) Main cluster for all corporate website applications.

Public Address: 177.0.2.56

The public address is required if the cluster front-ends an application.

☐ Process requests for unmapped hosts

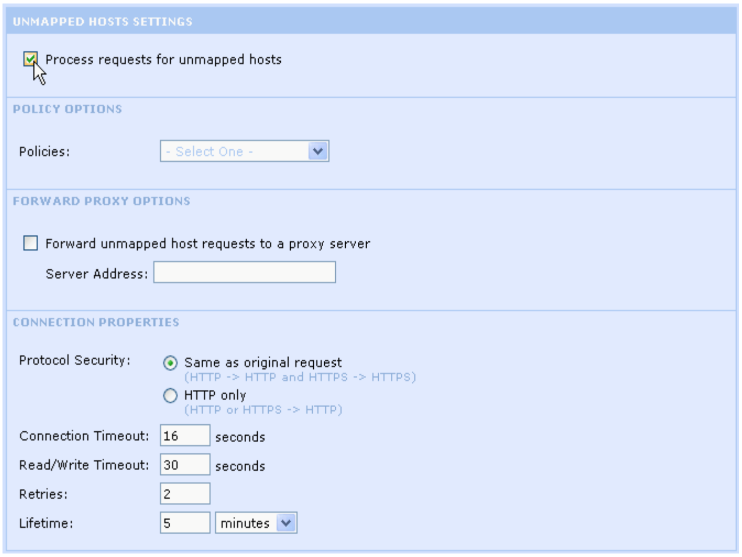
4. Provide a name for this cluster. The name should be unique in this WebAccelerator installation and be easily identifiable.
5. Describe the cluster and provide any pertinent information. For example, you might provide the location of the WebAccelerators you expect to be in this cluster or which applications you plan to have this cluster accelerate.
6. Provide the public address of the cluster. This should be the same as the IP address or hostname that you use to configure DNS so that requests can be directed to this cluster. This address is used by other WebAccelerators to route request. See 1 in Figure 4 Request Flow on page 24.

   If this cluster is not accelerating any applications, for example, if it is located at a remote user site and is only accelerating user requests back to the main clusters handling applications, you do not need to specify the public address.

Step 3 Unmapped Hosts Settings

7. Check this box if you want the cluster to handle requests to hosts that are not included in the host maps for any of the applications. See 2 in Figure 4 Request Flow on page 24. If you do not check this box, the cluster returns an HTTP 403 response code to the requesting client. If you check this box, more options appear.

   Warning: Before checking this box, be sure you understand the security implications. See "Security Implications" on page 22.


UNMAPPED HOSTS SETTINGS

☑ Process requests for unmapped hosts

POLICY OPTIONS

Policies:	- Select One -

FORWARD PROXY OPTIONS

☐ Forward unmapped host requests to a proxy server

Server Address:

CONNECTION PROPERTIES

Protocol Security:	● Same as original request (HTTP -> HTTP and HTTPS -> HTTPS) ○ HTTP only (HTTP or HTTPS -> HTTP)
Connection Timeout:	16 seconds
Read/Write Timeout:	30 seconds
Retries:	2
Lifetime:	5 minutes

8. Under Policy Options, select which set of policies you want applied to unmapped requests. See 4 in Figure 4 Request Flow on page 24.
9. Under Forward Proxy Options, check the Forward box if you want all the unmapped host requests to be forwarded to a proxy server, and provide the address of the proxy server. See 3 in Figure 4 Request Flow on page 24.
10. Use the default settings for connection properties or change them as appropriate.

    These are the same properties that you set for Destination Hosts. For more information on setting these values, see "Defining Destination Hosts" on page 32.
11. Click the Save button to save the cluster definition. You are taken back to the main cluster screen.

Step 4 Adding WebAccelerators to your clusters

12. The new WebAccelerators that are currently unassigned to a cluster are listed at the top of the table. Find the WebAccelerator you want to assign to your new cluster, and in its table row, select the cluster name from the drop-down cluster list.
13. Repeat, until all the WebAccelerators you want in that cluster are assigned to that cluster.
14. Click the Save button.

15. Click the Refresh button. In a few moments, you should see the WebAccelerator listed in the table under the cluster. Its status should now be `OK` instead of `Standby`, meaning it is active and can begin to handle requests for applications.
16. If needed, repeat Step 3 through Step 15 to create all the clusters you want, define them, and add the appropriate WebAccelerators to them. At any time, you can add or change your clusters, and move WebAccelerators between clusters.

Creating an Application Profile

The application profile provides key information to the WebAccelerator so that it can handle requests to your application appropriately.

Before you create an application profile, be sure that the actual physical Accelerators that you want to handle the application's requests are installed and assigned to a cluster. Use the Clusters screen to assign the Accelerators to a cluster, if you have not already done so.

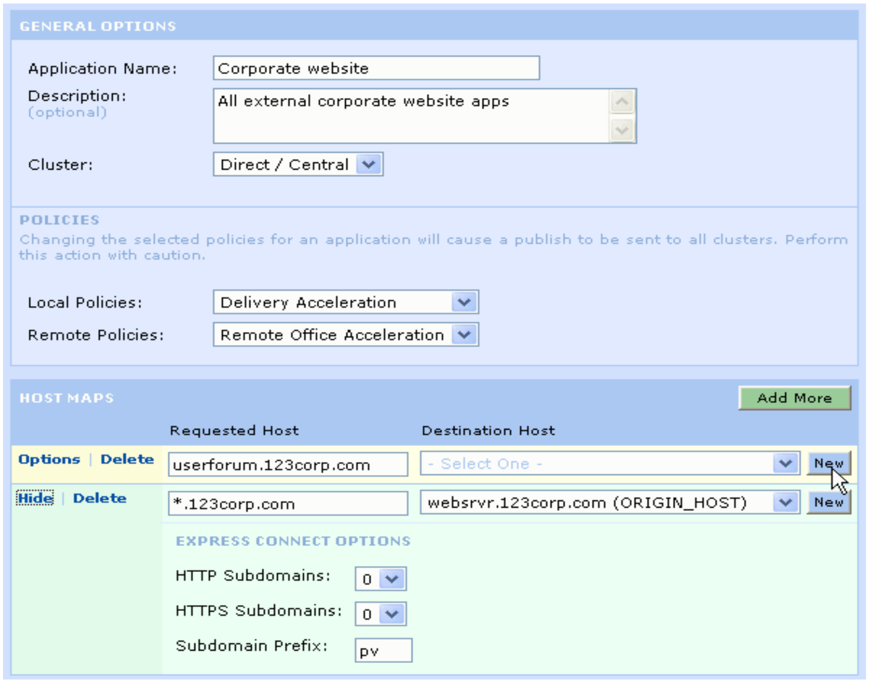
You might want to define your destination hosts (describe the origin servers this application is hosted by) before you begin your application profile. If so, go to the main Application screen and click on the Edit Destination Hosts link. See "Defining Destination Hosts" on page 32 for instructions.

Step 1 Access Applications screen

1. Log into the Admin Tool.
2. Click on the Applications menu item.
3. Click on the Create New Application link.

Step 2 General Options for your application

4. Provide a name for this application profile. The name should be unique in this WebAccelerator installation, and reflect the actual application at your site that you are accelerating.
5. Describe the application and provide any pertinent information. For example, you might provide the location of your servers that host this application, whether it is an internal or external site, who the users are, or any other useful information.



GENERAL OPTIONS

Application Name:	Corporate website
Description: (optional)	All external corporate website apps
Cluster:	Direct / Central

POLICIES

Changing the selected policies for an application will cause a publish to be sent to all clusters. Perform this action with caution.

Local Policies:	Delivery Acceleration
Remote Policies:	Remote Office Acceleration

HOST MAPS

	Requested Host	Destination Host	
Options | Delete	userforum.123corp.com	- Select One -	New
Hide | Delete	*.123corp.com	websrvr.123corp.com (ORIGIN_HOST)	New

EXPRESS CONNECT OPTIONS

HTTP Subdomains:	0
HTTPS Subdomains:	0
Subdomain Prefix:	pv

6. Select the cluster of WebAccelerators that are to handle requests for the application. Choose the cluster from the drop-down list. If the cluster is not listed there, go back to the main Cluster screen and define the cluster. This links the cluster to the application and its host map, and tells all the WebAccelerators in the cluster that they are accelerating this application. See a in Figure 4 Request Flow on page 24.

Step 3 Policies

7. Select the local policy set to be used by the cluster when handling requests for this application. See b in Figure 4 Request Flow on page 24. The policy set you select includes all policies, the Request Type Hierarchy, and any rewrite scripts. You can select one of the pre-defined policy sets included with the product, or you can select a custom policy set that you created.

   Select the policy set from the drop-down list and click OK. If you want to use a custom policy set but have not yet created one, select the standard policy set and use that until you have created and tested your own custom set.

8. Select the remote policy set to be used for this application. Generally, you use the default value for this option, the Remote Office Acceleration policy set. This policy set was designed to be used by any WebAccelerator outside the cluster when handling requests for this application.

   In most cases, if any WebAccelerator outside the cluster receives a request for this application, it forwards the request to an WebAccelerator that is part of the cluster. The set of policies you specify in this option only applies to requests such as these that are being forwarded. See c in Figure 4 Request Flow on page 24. Once the request is received by the application's cluster, the policy set you specified in Step 7 is used to handle the request.

   If you have a remote office with an WebAccelerator, generally any request a user in that office makes is directed to that WebAccelerator. If the Accelerator cannot service the request from its own cache, it forwards the request to the appropriate cluster, after applying the policies you set here. These policies usually specify options such as compression to speed the transfer, and add a special header that includes the IP address of the originating client of the request.

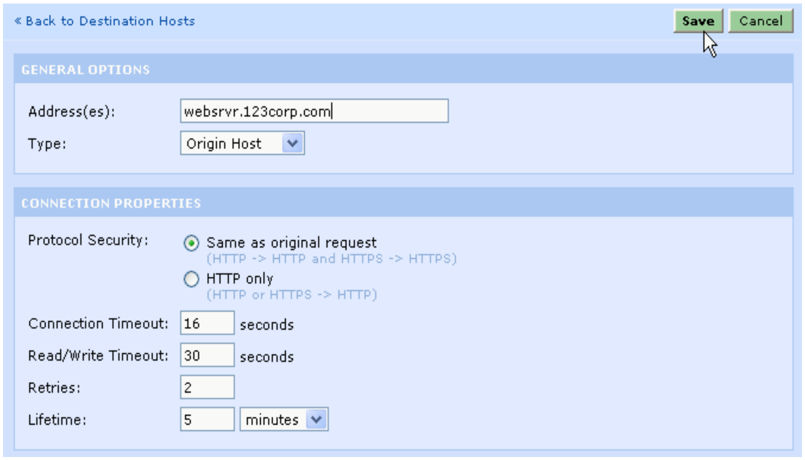
Step 4 Host Maps

Perform these steps for every domain for which your site expects to receive requests. You must also provide a domain mapping for every domain referenced in your esi:include tags. ESI and esi:include are described in Chapter 9, Assembling Content with ESI. For information on unmapped hosts, see "Unmapped Hosts" on page 21.

9. Specify the Requested Host. The Requested Host is the hostname you expect to see in HTTP requests for this application. For information on the valid hostnames you can specify, see "Valid Hostnames" on page 22. The Requested Host value is what the HOST header in all requests is compared to. See d in Figure 4 Request Flow on page 24.

10. Select the Destination Host from the drop-down list. The Destination Host is the origin host to which the WebAccelerator proxies for fresh content. See e in Figure 4 Request Flow on page 24. The Destination Host should be the DNS name that you use for your origin servers once the WebAccelerator is put into production.

    If the Destination Host you specify here is new, click the New button to the right of the Destination host name. You can now define this host to the system. For example, suppose your site is normally visited using the domain `www.123corp.com`. Also, suppose your origin servers are being moved to `ows.123corp.com`. Enter `www.123corp.com` under Requested Host and click the New button for Destination Host. On the Destination Host screen, fill out the information and click the Save button:



Address(es):	websrvr.123corp.com
Type:	Origin Host
Protocol Security:	[✓ Same as original request (HTTP -> HTTP and HTTPS -> HTTPS)] HTTP only (HTTP or HTTPS -> HTTP)
Connection Timeout:	16 seconds
Read/Write Timeout:	30 seconds
Retries:	2
Lifetime:	5 minutes

For information on the options to set for Destination Host, see "Defining Destination Hosts" on page 32.

11. To specify Express Connect options for connections to this host, click on the Options link for this host. Set these options before you save the host map. Click Hide if you want to collapse the area. See "Managing Express Connect" on page 34 for information about the options.
12. Click the Add More button to add another line for Requested Host and Destination Host if you have another Requested Host you need to map to an origin server.
13. Repeat Step 9 to Step 12 until you have completely mapped all potential domains that might appear in a request to this application.
14. Click the Save button.

Your application profile is now active and in use by the WebAccelerator.

Invalidation Service Password

From the Applications screen, you can set your invalidation service password. The invalidation service password is used only for ESI invalidation requests. The default password is `invalidator`. Click on the link to change the password.

Defining Destination Hosts

When you define a Destination Host, you must specify the Host Address and Host Type options. If you do not specify new values for the connection properties, the default values are used.

Host Address

Specify the DNS name of the server that the WebAccelerator should use when proxying for new or fresh content. If you are specifying the name for an origin server, you can include the origin server ports for HTTP and HTTPS:
`hostname:insecure:secure`

where *`hostname`* is the address for the origin server, such as `main.123corp.com`, *`insecure`* is the port number for the HTTP port, and *`secure`* is the port number for the HTTPS port. The HTTP port is always shown first. You do not need to include the HTTPS port. If you want to specify an HTTPS port, you must include the HTTP port. These are valid host addresses:

```
main.123corp.com:8000:8443
main.123corp.com:8000
123corp.com:80:443
123corp.com
```

The last two host addresses are equivalent.

To implement load balancing, you can enter a range of servers here, delimited by a slash (/), or a list of servers delimited by a comma (,), or some combination of both. If you specify a range, it must take the form of names with a numerical value at the end of a subdomain name. For example, these are valid examples of ranges:

```
ows4.swanlabs.com/ows20.swanlabs.com
ows.appserver2.swanlabs.com/ows.appserver50.swanlabs.com
```

If a particular host is unavailable, the WebAccelerator does not use that hostname in its load balancing until that host is available.

By default, the WebAccelerator sets a cookie to ensure a given client's connections are always sent to the same origin server. This capability can be managed using the appropriate `pvsystem.conf` parameters. See the *Administration Guide* for information on changing the `pvsystem.conf` file.

Host Type

Indicate whether the destination host is a normal origin server or a proxy server. If `Proxy Server` is selected, then all proxies to this server are performed in a way that is compatible with a proxy server.

For example, the GET field in the request might be changed from:

```
GET /apps HTTP/1.1
```

to:

```
GET http://www.123corp.com/apps HTTP/1.1
```

Security

Specify the protocol to be used for connections between the WebAccelerators and your origin servers. If you choose Same as Original Request, then the protocol used for the client request is used for proxies to your origin servers. If you choose HTTP, then only HTTP is used for proxies even if HTTPS is used for the original request. Select HTTP only if your WebAccelerators and your origin servers are on the same protected network and you want to avoid the overhead of HTTPS connections.

Connection Timeout

Specify the time in seconds your WebAccelerators should wait for a response from your origin servers when proxying for content.

Read/Write Timeout

Specify the time in seconds your WebAccelerators should wait for a read or write operation to your origin servers to complete before giving up.

Retries

Specify the number of times your WebAccelerators should attempt to connect to your origin servers in the event of a connection failure.

Lifetime

Specify the time your WebAccelerators should hold a persistent connection open to your origin servers. You specify a number and select the unit of time (minutes or seconds) to be used.

Each time an WebAccelerator first proxies to an origin server, it forms a persistent connection to that server. All subsequent proxies are performed over that connection. However, some web servers and application servers timeout persistent connections regardless of the activity occurring on the connection. Therefore, before performing each proxy, the WebAccelerator checks to see if the connection has been opened for a longer period of time than is defined on this field. If so, the WebAccelerator drops the connection and reconnects to your origin server before attempting the proxy.

Managing Express Connect

Express Connect options allow you to identify subdomains that the WebAccelerator automatically generates for qualifying links and embedded URLs. Most browsers create up to two persistent TCP connections for each domain from which they are requesting data. By modifying embedded URLs with unique subdomains, the WebAccelerator is able to convince the browser to open more persistent connections (up to two per subdomain generated by the WebAccelerator). The result is that the browser downloads data much faster and the user perceives a much faster response time from your site.

To use Express Connect, identify the number of subdomains you want the WebAccelerator to generate for each supported protocol (HTTP or HTTPS). Also, identify the subdomain prefix that you want the WebAccelerator to use:

- for the number of subdomains, select the maximum number of subdomains that you want the WebAccelerator to generate for each page that it serves. Remember that for each additional subdomain you request, the browser probably opens two additional persistent connections over and above what it normally does. If you request 1 subdomain for the HTTP protocol, a browser could open up to a total of 4 persistent connections to the WebAccelerator when requesting pages over the HTTP protocol.

  Note that the WebAccelerator uses these additional subdomains only on embedded URLs or links that request images or scripts.

- for the subdomain prefix, select the prefix that you want used for your subdomains. For example, suppose the Requested Host for this mapping is `www.123corp.com` and you request 2 additional subdomains for the HTTP protocol. Further, suppose you use a Subdomain Prefix of `pv`. In this case the WebAccelerator changes the domain on qualifying embedded URLs and links to use these domains:

  ```
  pv1.www.123corp.com
  pv2.www.123corp.com
  ```

  You must configure DNS with these entries and they must map to the same IP address as your base origin server (`www.123corp.com` in this example). Note that these domains are used only for requests/responses between the client and the WebAccelerator. Your origin servers never see a request that uses these domains.

Chapter 4

Managing Policies

What Is a Policy Set?
Managing Policy Sets
Using the Policy Editor
Publishing Policies
HTTP Request Data Type Parameters

All policies are based on the information the WebAccelerator finds in an HTTP request. For example, a policy can be based on whether or not a request contains a particular cookie. If it contains the cookie, the policy can specify that the request is proxied to the origin servers and the response is never cached. The types of information in an HTTP request that the WebAccelerator can analyze when applying policies are called the HTTP Request Data Type Parameters.

You create and manage policies using the Admin Tool. You can view any policy in the Admin Tool. You can only modify policies in custom policy sets. The pre-defined policy sets shipped with the WebAccelerator cannot be changed, only replaced by newer versions. To modify the policies in a pre-defined policy set, copy the set. The copy is a custom policy set and you can modify it as you like.

Policies you create or modify are not made available to applications until they are published. The WebAccelerator maintains a development copy of each policy set, and creates a production copy when you publish the policy set. If you are editing policies on a published policy set, you are actually making the changes to the development copy of the set. The production copy is in use by applications. You can make as many changes as you like and not affect any applications until you publish your changes. Then the changes are propogated to the production copy and start being used by the WebAccelerator to handle requests.

For most policy management, log into the Admin Tool and click on Policies. Clicking on the Edit or View link for any policy set takes you to the Policy Editor. Detailed information on the individual screens in the Admin Tool is provided in the online help available as part of the user interface.

What Is a Policy Set?

A policy set is the collection of rules and scripts that determine how each request is handled and how the content for responses is cached and assembled. Each policy set consists of:

- matching policies
- a Request Type Hierarchy, which is determined by the matching policies
- caching policies
- proxy policies
- response assembly policies, including content variation and substitution
- lifetime and invalidation trigger policies, which control when certain content is tagged as invalid (out of date) or expired and should be refreshed
- connection mapping policies
- rewrite scripts, for post-processing responses received from the origin servers before caching and sending the response
- log formats, to enable you to customize WebAccelerator log information for better system analysis

Most of these policies are based on information present in the request. Matching policies look at the information in the request to group requests. For example, they might place all requests that have a `.gif` extension into a group for images. A response assembly policy might look at a query parameter in a request and use its value to substitute for a value in the cached response before assembling the page. For more information on the types of information the policies are based on, see "HTTP Request Data Type Parameters" on page 45.

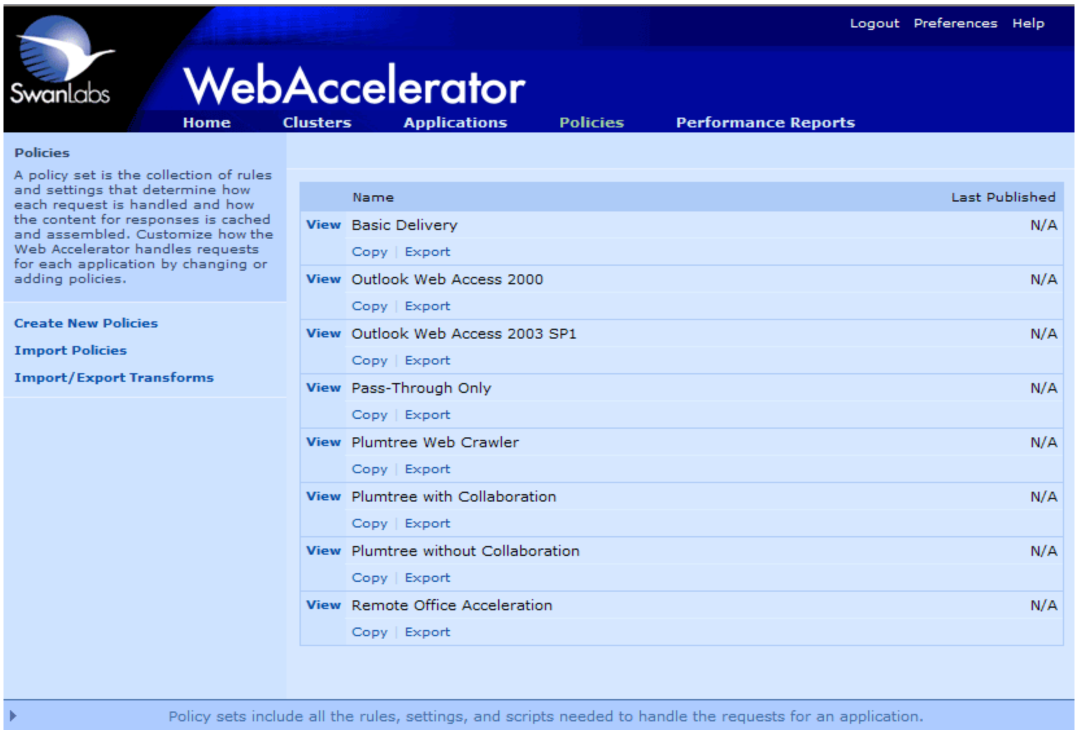
Any particular policy can be quite unique to an application. An auction site needs invalidation trigger policies that invalidate cached content for an item whenever a new bid comes in. A site with rotating advertisements needs a random variation policy to change the ads that appear on a page when it is assembled.

Rather than assigning, for example, 50 out of 1000 miscellaneous policies to an application, you design a complete set of policies for each application, containing at least one policy of each type (caching, proxy, lifetime, and so on). If your site does not require a unique set, you can use one of the pre-defined policy sets that ship with the WebAccelerator. If you have two very similar applications, you can use the same policy set for both applications. However, for simplicity, you always modify policies as part of a particular policy set.

Managing Policy Sets

When you install the WebAccelerator, several pre-defined policy sets are installed that you can use for your applications. The pre-defined policy sets are designed to work well in most cases, though you might need to tailor one specifically for your application to achieve the highest level of performance for your system.

To view these pre-defined policy sets, log into the Admin Tool and click on the Policies item in the functional menu bar. The darker rows at the top of the table contain the pre-defined policy sets. Until you create some custom sets, these are the only policy sets in the table. Click on the View link to explore a policy set and look at the individual policies and their settings.



Policies

A policy set is the collection of rules and settings that determine how each request is handled and how the content for responses is cached and assembled. Customize how the Web Accelerator handles requests for each application by changing or adding policies.

	Name	Last Published
View	Basic Delivery Copy | Export	N/A
View	Outlook Web Access 2000 Copy | Export	N/A
View	Outlook Web Access 2003 SP1 Copy | Export	N/A
View	Pass-Through Only Copy | Export	N/A
View	Plumtree Web Crawler Copy | Export	N/A
View	Plumtree with Collaboration Copy | Export	N/A
View	Plumtree without Collaboration Copy | Export	N/A
View	Remote Office Acceleration Copy | Export	N/A

Policy sets include all the rules, settings, and scripts needed to handle the requests for an application.

To tailor one of these pre-defined policy sets:

1. Click on the Copy link under the policy set. A dialog box appears.
2. Provide a name for this policy set that is descriptive of the tailoring you plan to do, and also provide a brief description.

3. When you are satisfied with the name and description, click the Copy button. The new policy set should appear as a new row in the table.
4. To tailor this policy set, click on its Edit link. You are taken to the Policy Editor, where you can edit any of the policies or delete or add new policies.

For more information about editing or creating policies, see "Using the Policy Editor" on page 40.

For any of the tasks in the next sections, start by logging into the Admin Tool and clicking on Policies in the function menu to go to the main Policies screen.

Creating Policy Sets

Click on the Create New Policies link. A dialog box opens. Enter a name and description for your new policy set. Make the name descriptive so you can easily tell the purpose of this policy set whenever you see the name.

Click on the Create button to save the name and description. A new row is created in the Policy table for the policy set. You have created the policy set container. Now you need to populate it with actual policies. Click on the Edit link next to the policy set in the table. You are taken to the Policy Editor where you can start creating policies.

Copying Policy Sets

To copy a policy set, click on the Copy link under the policy set you want to copy. A dialog box opens. Enter a name and description for the copy you are creating. Make the name descriptive so you can easily identify this policy set whenever you see the name.

Click on the Copy button to save the name and description. A new row is created in the Policy table for the policy set. You have created a new policy set that contains a copy of everything in the original policy set. To make changes, click on the Edit link next to the policy set in the table. You are taken to the Policy Editor where you can start your modifications.

Renaming Policy Sets

To rename a policy set, click on the Rename link under the policy set you want to rename. A dialog box opens. Enter the new name and description.

Click on the Rename button to save the name and description. The policy set is renamed and the new name should appear in the table.

Deleting Policy Sets

To delete a policy set, click on the Delete link under the policy set you want to remove. A dialog box opens and asks if you are certain you want to delete this policy set. Click on the OK button to delete the policy set. The policy set and everything in it is deleted.

There is no recovery, so don't delete a policy set unless you are sure you do not want to refer to it or look at any of its policies ever again. You can keep a policy set to refer to, even if there is no application using it.

Exporting Policy Sets

Exporting a policy set is different than copying it. Use Export to back up or transfer policy sets to another machine. For example, if Swan Labs Customer Confidence asks you for a copy of your policies, you export a copy and send that. If you want to copy a policy set to a separate WebAccelerator system, you export the policy set.

Exporting a policy set creates an XML file that can be imported into any WebAccelerator system that is at the same version level.

Click the Export link under the policy set. A dialog box opens. Choose whether you want to export the published or development version of the policies. The published version is the one currently in use by the system. The development version includes policies in the set that are not yet published. Click the Export button.

The XML export file is created and displayed in a separate browser window. Use the browser Save As... function to name the file and save it on your desktop.

Backup: Swan Labs recommends using Export to back up your policy sets on a regular basis and whenever you publish changed policies. After exporting the set, you can place the XML file in your backup and archival system.

Importing Policy Sets

You can only import policies that have been exported. Importing a policy set also publishes the set. Before you start, be sure there is an XML export file available to your desktop for the policy set you want to import.

Click the Import Policies link. A dialog box appears. Click the Browse button to browse and locate the XML file you want to import.

If you want to replace an existing policy set with an imported policy set that has the same name, check the `Overwrite existing policy of the same name` box. If you do not want to replace an existing policy set, leave the box unchecked and the name of the imported policy set is modified with a copy number.

You can use Rename after import to change the name of the imported policy set.

Importing and Exporting Transforms

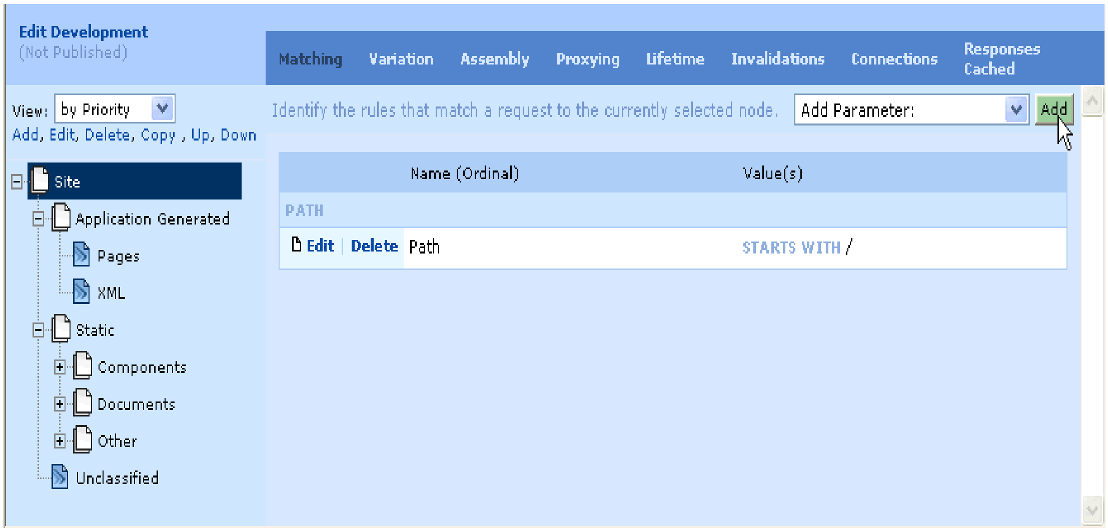
Transforms are the rewrite engine scripts that can be used to modify a response before it is cached and sent to the client. They are stored as a zip file and can be exported to your desktop for modification and then imported back into the WebAccelerator. For more information, see "Customizing Rewrite Scripts" on page 196.

Using the Policy Editor

The Policy Editor is part of the Admin Tool. You use the Policy Editor to view and edit individual policies inside a policy set. First find the policy set on the main Policies screen, and click the View or Edit link to go to the Policy Editor. If you want to switch to a different policy set, close the Policy Editor and click the link for the next policy set you want to view or edit.

There are two main menu bars in the Policy Editor user interface:

Figure 5 Example of Policy Screen in Policy Editor



Identify the rules that match a request to the currently selected node.

	Name (Ordinal)	Value(s)
PATH		
Edit | Delete	Path	STARTS WITH /

- the function menu bar at the left of the screen under the Swan Labs logo, which takes you to:

  - Policies, where you view and change your individual nodes and policies
  - Log Format, where you can select different log formats, described in Chapter 15, Setting Log Formats
  - Publish, where you publish your policies, which makes them available to your applications for use, described in "Publishing Policies" on page 45
- the policy menu bar in the center of the screen , which takes you to:
  - Matching, where you define matching policies, described in Chapter 6, Setting Matching Policies
  - Variation, where you define variation policies, described in Chapter 7, Setting Variation Policies
  - Assembly, where you define assembly policies, described in Chapter 8, Setting Assembly Policies
  - Proxying, where you define proxying policies, described in Chapter 10, Setting Proxying Policies
  - Lifetime, where you define cache lifetime policies, described in Chapter 11, Cache Control and Content Lifetime
  - Invalidations, where you define invalidation triggers used for refreshing content, described in Chapter 12, Invalidating Cached Content
  - Connections, where you define connection mapping policies, described in Chapter 13, Connection Mapping
  - Responses Cached, where you define caching criteria for responses, described in Chapter 14, Setting Responses Cached Policies

Policy Editor Basics

This section discusses the basics of using the Policy Editor. These principles are fundamental to the Policy Editor, but to fully understand them, you need to understand the Request Type Hierarchy, application matching, and the different types of policies and how you create them. After reading those sections of this guide, you might want to reread this section.

Editing Mode

If you want to change a policy, be sure you are in edit mode. Click on Edit Development after you are in the Policy Editor to switch to edit mode. If you are changing a published policy set, you must republish it before your changes take effect.

Select Node First

Always start by selecting the node. Each policy is associated with a node in the Request Type Hierarchy. Before you can view, add, or edit a policy, you must select its node. For more information, see Chapter 5, The Request Type Hierarchy.

Select Type of Policy Next

To find the policy you want to view, add, or edit, select the link for the type of policy you want to see. All policies for a node are organized under links by type. For example, to see the matching policies in effect for a node, click on the Matching link.

Add by Selecting Parameter Type

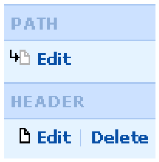
When adding a policy or rule, always select the parameter type for the rule first, then click the Add button. Because each parameter type is specified a different way, the Add screen varies for each. Selecting the parameter type from the drop-down Add Parameter list first tells the Policy Editor which Add screen to display when you click the Add button.

If a policy has no rules currently specified for it and there is no parameter drop-down list, click the Add button to go to a screen where you can create a rule.

Understand Inheritance for the Node

Before editing an existing policy, be sure you understand which nodes the policy is inheriting its rules from, and decide whether you want to change the rules at this node or whether you want to change the rules at an ancestor node.

The nodes that policies are associated with are organized in a tree hierarchy, and each leaf node can inherit policies and parameter rules from its parents and grandparents and so on. This is explained in detail in Chapter 5, The Request Type Hierarchy. Whether a particular rule is inherited is indicated in the policy details. For example:

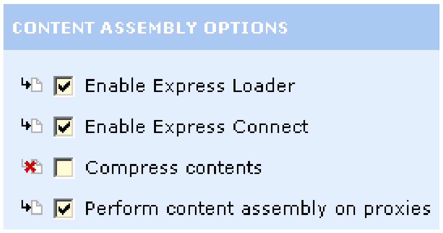


The page icon with the downward arrow indicates the Path parameter rule is inherited from an ancestor node, while the Header parameter rule applies only to this node and is not inherited. Note that you can remove the Header parameter rule from this policy, but not the Path parameter rule. To remove an inherited rule, you must find the ancestor node for which the rule is specified, and remove it there. You might need to add the rule back into other child nodes, because removing an inherited rule removes it from all the children of that node, even ones for which the rule should still apply.

Determine Where to Edit an Inherited Rule

Once you find that a parameter rule is inherited, you choose between editing that rule at the node, which overrides the inherited values for the rule, or changing the rule at one of the ancestor nodes. Be sure you understand the inheritance chain for the node before you begin editing rules at its ancestors. For example, in this assembly policy, at the leaf node, the existing rules are:

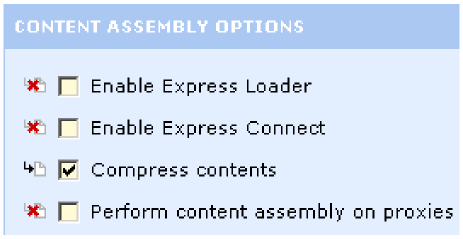
Figure 6 Leaf node, which is a child


CONTENT ASSEMBLY OPTIONS

- [✓] Enable Express Loader
- [✓] Enable Express Connect
- [ ] Compress contents
- [✓] Perform content assembly on proxies

no options are enabled, and these settings are inherited, but the `Compress contents` setting as been overridden. The page icon with the red slashed circle indicates a rule was inherited but changed at the current node (overridden). At the parent node, the other settings must have been unchecked, but `Compress contents` was checked, and at this leaf node, it must have been unchecked, to override it. If we look at the parent node's settings:

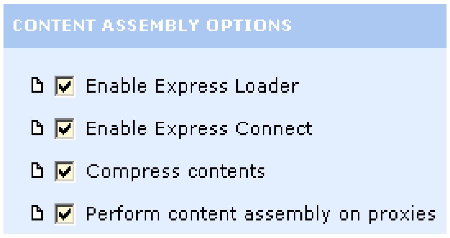
Figure 7 Parent of leaf node


CONTENT ASSEMBLY OPTIONS

- [ ] Enable Express Loader
- [ ] Enable Express Connect
- [✓] Compress contents
- [ ] Perform content assembly on proxies

we see this is correct, but we also see that the other settings were overridden here, so they must have been checked at the parent's parent node. If we look at this grandparent node,

Figure 8 Grandparent of leaf node


CONTENT ASSEMBLY OPTIONS

- [✓] Enable Express Loader
- [✓] Enable Express Connect
- [✓] Compress contents
- [✓] Perform content assembly on proxies

we see that all the options are actually set here, not inherited from any other node, and they are all enabled.

If, for example, you want to enable the Express Loader at the original leaf node, you must decide if you want to:

- override the inherited setting at the leaf and check the box, or
- cancel the override setting at the parent, so that the parent inherits the Express Loader enabled setting of the grandparent, and in turn passes that setting to the leaf node

If you cancel the override setting at the parent, keep in mind that this action changes the settings for all the children of this parent, not just the leaf node you originally wanted to change.

This sounds complex, but if you set up your policies and nodes in a thoughtful and logical way, you should only specify rules for ancestor nodes that you want most of the child nodes to inherit. If you have a rule set at a node that is overridden at most of the offspring nodes, perhaps the rule should not be set at that node, but instead should be set only at the offspring that use the rule.

Parameter Restrictions

Rules based on parameters are mainly restricted by the way the parameter appears in an HTTP request. For example, for each policy, you can only have one rule for the Method parameter, because a request can only have one method, such as GET. You can have multiple rules for Query Parameter, because a request might have several query parameters present in it. Rules for named Query Parameter must specify the name of the query parameter. Rules for Unnamed Query Parameter must specify the location of the query parameter in the request. These sorts of restrictions make sense when you understand the structure of HTTP requests.

Publishing Policies

Policies that you created using the Policy Editor are published to make them available for use by applications. Once a policy set is published, you can select it for an application.

Publishing is a simple process that is performed by selecting the Publish link in the Policy Editor. For details on how to publish your policies, see the online help for the Publish screen.

You can also use the Publish screen to perform these activities:

- enable debugging

  There is a debugger that helps you understand how your ESI tags are affecting the content served by the WebAccelerator.

- override compression

  The WebAccelerator is capable of compressing the content that it serves for you using gzip-encoding. This encoding is helpful to reduce bandwidth costs for your site. It also improves your site's performance as perceived by your users, such as those connected using dial-up. However, gzipping data served by the WebAccelerator can make it difficult to debug the WebAccelerator's behavior in some circumstances. Therefore, you can override any gzip-encoding policies set in the published policy set.

  For information on managing gzip-encoding of the data served by the WebAccelerator, see "Compressing Content" on page 88.

HTTP Request Data Type Parameters

Primarily the WebAccelerator performs its activities by examining information on an HTTP request URL and matching that information to caching policies you define. The caching policies, in turn, might be based on the information available in the HTTP request URL.

Before the WebAccelerator attempts to match the information in the URL to values specified in policies, it first translates any escaped characters, such as `%2F` or `%2E`, back to their regular form, such as `/` or `.`, for matching.

It is important to understand the HTTP request data types on which the WebAccelerator can base its activities. In the WebAccelerator, these data types are used as parameters that you can specify to define application matching and caching policies.

This table briefly identifies the HTTP request data type parameters used by the WebAccelerator and what types of policies use that particular parameter type:

Table 1 HTTP Request Data Parameters

Parameters	Matching	Variation	Assembly	Proxying	Invalidations	Connections
Protocol	x	x		x	x	x
Host	x	x		x	x	x
Path	x				x	x
Extension	x				x	x
Method	x	x		x	x	x
Query Parameter	x	x	x	x	x	x
Unnamed Query Parameter	x	x	x	x	x	x
Path Segment	x	x	x	x	x	x
Cookie	x	x		x	x	x
User Agent	x	x		x	x	x
Referrer	x	x		x	x	x
Header	x	x		x	x	x
Client IP	x	x		x	x	x

To learn how to use these parameters to create policies, read:

- Chapter 5, The Request Type Hierarchy
- Chapter 6, Setting Matching Policies
- Chapter 7, Setting Variation Policies
- Chapter 8, Setting Assembly Policies
- Chapter 10, Setting Proxying Policies
- Chapter 12, Invalidating Cached Content
- Chapter 13, Connection Mapping

The other types of policies, described in Chapter 11, Cache Control and Content Lifetime and Chapter 14, Setting Responses Cached Policies, do not use these parameters in their rules.

Specifying Parameters

In general, parameters are specified in policies for the purpose of matching information in a request to that parameter. If there is a match, the WebAccelerator behaves a certain way, such as proxying the request or substituting a value in the response. For certain types of policies, you can specify what the behavior should be. For others, there is no choice because the behavior is not configurable. In some cases when you specify a parameter, you can specify whether the WebAccelerator behavior is triggered by the request matching the parameter or by the request not matching the parameter.

Note: If you specify that an action is triggered by the request not matching a certain value for a parameter, that action is triggered if the parameter in the request is a different value or is empty (null). The action is not triggered if the parameter does not appear in the request.

There are three main types of information you provide when you specify a parameter:

1. information to identify the parameter in a request: the parameter type, and possibly the name of the parameter or location of the parameter in the request
2. information about what value or state the parameter in the request should have for a match to occur:
   - that the parameter is present in the request and it matches the specified value, provided in the form of a regular expression, or
   - that the parameter is present in the request and it does not match the specified value, provided in the form of a regular expression, or
   - that the parameter is present in the request but has no value (is an empty string), or
   - that the parameter is not found in the request at all
3. information about the actions the WebAccelerator takes, if this option exists for the parameter and policy you are specifying:
   - whether the action occurs on a match or when there is no match
   - what action or behavior to take, which varies depending on the type of policy

When you specify a parameter in a policy, you begin by choosing the type of parameter, such as Cookie or Extension. The way you specify a parameter is dependent on the type. For example, to identify a Protocol parameter for matching, you only need to check a box to indicate whether the protocol for a matching request needs to be HTTP or HTTPS. To identify a Cookie parameter, you need to provide a name in order to identify the cookie, in addition to the value or state.

In the Policy Editor user interface, you select the parameter type and click the Add button. You are taken to a screen that prompts you for the information required for this specific policy and parameter. See the following sections on each parameter type to learn more about identifying parameters and specifying values for them. See the chapters on the different policy types to learn more about the WebAccelerator actions associated with each type of policy.

Protocol

Policies that use the Protocol data type are based on the protocol used for the request. You can define caching behavior based on the HTTP and HTTPS protocols.

For example, the URL:
`http://www.swanlabs.com/apps/srch.jsp?value=computers`

uses the HTTP protocol.

Host

Policies that use the Host data type are based on the value provided for the HTTP HOST request header. This header describes the DNS name that the HTTP request is using. For example, the URL:
`http://www.swanlabs.com/apps/srch.jsp?value=computers`

results in this HTTP HOST request header:
`HOST: www.swanlabs.com`

Path

Policies that use the Path data type are based on the path portion of the URI. The path is defined to be everything after the host in the URL, and up to the end of the URL or the question mark (?), whichever comes first. For example, the paths for the URLs:

```
http://www.swanlabs.com/apps/srch.jsp?value=computers
http://www.swanlabs.com/apps/magic.jsp
```

are, respectively:

```
/apps/srch.jsp
/apps/magic.jsp
```

Extension

Policies that use the Extension data type are based on the value that follows the right-most period in the right-most segment key. For example, in these URLs, `gif`, `jpg`, and `jsp` are all extensions:

```
http://www.swanlabs.com/images/up.gif
http://www.swanlabs.com/images/down.jpg
```

`http://www.swanlabs.com/apps/psrch.jsp;sID=AAyB23?src=magic`

Segment keys (the text following the semicolon (;) and preceding the question mark (?) in the third URL) are described in "Path Segment" on page 50.

Method

Policies that use the Method data type are based on the method used for the request. You can define caching behavior based on the GET and POST methods.

Query Parameter

Policies that use the Query Parameter data type are based on a particular query parameter that you identify by name, and you provide a value to match against. Usually this is a literal value that must appear on the query parameter in the request, or a regular expression that must match the request's query parameter value. The query parameter can be in a request that uses either the POST or GET methods.

You can create a rule that matches when the identified query parameter is provided with an empty value or when it is absent from the request. For example, in this URL the `action` query parameter provides an empty value:
`http://www.swanlabs.com/apps/srch.jsp?action=&src=magic`

Unnamed Query Parameter

Unnamed query parameters are query parameters that have no equal sign (=). That is, only the query parameter value is provided on the request. For example, this URL includes two unnamed query parameters that have the value of `dog` and `cat`:
`http://www.swanlabs.com/apps/srch.jsp?dog&cat&src=magic`

Policies that use the Unnamed Query Parameter data type specify the ordinal of the parameter instead of a parameter name. The ordinal is the position of the unnamed query parameter in the query parameter portion of the URL. You count ordinals from left to right, starting with 1. In the above URL, `dog` is in ordinal 1 and `cat` is in ordinal 2.

You can create a rule that matches when the identified query parameter is provided with an empty value or when it is absent from the request. For example, in this URL, ordinal 1 provides an empty value:
`http://www.swanlabs.com/apps/srch.jsp?&cat&src=magic`

In this URL, ordinal 3 is absent (`dog` being in ordinal 1 and `src` being in ordinal 2):
`http://www.swanlabs.com/apps/srch.jsp?dog&src=magic`

Path Segment

The Path Segment data type is used to identify one of two values:

- the value provided for a specific segment key
- the value provided for a specific segment parameter

Segment Keys

In a path, a segment is the portion of a URI path that is delimited by a forward slash (/). For example, in the path:
`/apps/search/full/complex.jsp`

`apps`, `search`, `full`, and `complex.jsp` all represent path segments. Further, each of these values are also the segment key, or the name of the segment.

Segment Parameters

A segment parameter is a value that appears after the segment key in a path segment. Segment parameters are delimited by semicolons (;). For example, `magic`, `shop`, and `act` are all segment parameters for their respective path segments:
`/apps/search/full;magic/complex.jsp;shop;act`

Segment Ordinals

To identify a segment, you must provide an ordinal that identifies the location of the segment in the path:
`/apps/search/full;magic/complex.jsp;shop;act`

You must indicate how you are counting within the path: from the left or the right (always count starting from 1). For the example used here, `/full;magic` has these ordinals depending on how you are counting:

Ordinal	Numbering Selection
3	Numbering Left-to-Right in the Full Path
2	Numbering Right-to-Left in the Full Path

Segment Parameter Ordinals

Once you have identified a segment's ordinal, you must identify the ordinal of the element within the segment that the policy is based on. Counting is always left-to-right, and the segment key is always ordinal 0. For example, in this segment:
`/complex.jsp;shop;act`

The segment key is `complex.jsp` and its ordinal is 0. Segment parameters `shop` and `act` have ordinals 1 and 2 respectively.

Cookie

Policies that use the Cookie data type are based on a particular cookie that you identify by name, and you provide a value to match against. Usually this is a literal value that must appear on the cookie in the request, or a regular expression that must match the request's cookie.

The names of the cookies that you identify must match the names that appear on the COOKIE HTTP request headers. These are also the same names you used to set the cookies using the HTTP SET-COOKIE response headers.

You can create a rule that matches when the identified cookie is provided with an empty string or when it is absent from the request. For example, here the `REPEAT` cookie is empty:

```
COOKIE: REPEAT=
```

Here, the `REPEAT` cookie is absent:

```
COOKIE: USER=334A5E4
```

User Agent

Policies that use the User Agent data type are based on the value provided for the HTTP USER_AGENT request header. This header identifies the browser that sent the request. For example, this USER_AGENT request header:

```
USER_AGENT: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
```

indicates that the requesting browser is IE 5.01 running on Windows NT 5.0.

Usually you do not base policies on the USER_AGENT request header unless your site behaves differently depending on the browser in use.

Referrer

Policies that use the Referrer data type are based on the value provided for the HTTP REFERER request header. (Note the misspelling of REFERER. This spelling is defined for this request header in all versions of the HTTP specification). This header provides the URL location that referred the client to the page that the client is requesting. That is, REFERER provides the URL that contains the hyperlink that the user clicked in order to request the page. For example:

```
REFERER: http://www.swanlabs.com/
```

Usually you do not base policies on the REFERER request header unless your site behaves differently in some way depending on who the referrer is. This is commonly used for sites that provide different branding for their pages based on the web portal or search engine that the user used to find the site.

Header

Policies that use the Header data type are based on a particular header that you identify by name, and you provide a value to match against. The Header data type is used to create rules based on any request header other than one of the recognized HTTP request data types listed in Table 1 HTTP Request Data Parameters on page 46. These can be standard HTTP request header fields such as AUTHORIZATION, CACHE-CONTROL, and FROM. They can also be user or application defined headers in the form of a structured parameter.

These are examples:

```
Accept: text/html, image/*
Accept-Encoding: gzip
Accept-Language: en-us
CSP-Gadget-Realm-User-Pref: NM=5,PD=true,R=3326
```

The last header in the example shows a structured parameter. If you specified this header as a structured parameter when creating your policy, it is parsed into name= value pairs when the request is being examined by the WebAccelerator. If you did not specify it as a structured parameter, it is treated like a normal header, where everything after the colon (:) is treated as the value.

The format of a structured parameter in a request is similar to that used for Cookie, with a header name that you choose, followed by a series of name=value pairs separated by commas. The header name is not case-sensitive. In this structure, semicolons (;) are special characters. Anything after a semicolon is ignored by the parser until it reaches the subsequent comma. For example, these are valid header structured parameters:

```
CSP-Global-Gadget-Pref: AT=0
CSP-Gadget-Realm-User-Pref: NM=5,PD=true,R=3326
CSP-User-Pref: E2KU=chi,E2KD=ops%u002eswanlabs%u002enet,E2KM=chi
CSP-Gateway-Specific-Config: PT-User-Name=chi,PT-User-ID=
212,PT-Class-ID=43
Standards: type=SOAP;SOAP-ENV:mustUnderstand="1",version=1.2
```

In the last line, `SOAP-ENV:mustUnderstand="1"` is ignored because it follows a semicolon, but `version=1.2` is read as a name=value pair because it follows a comma. If you have metadata to include in the header that you want the WebAccelerator to ignore, put it after a semicolon.

To specify a header as a structured parameter when you are creating or editing a policy, you specify the name using this syntax:

`headername``:`*`parmname`*

where:

`headername`	is the name of the header.
`parmname`	is the name of the parameter whose value you want to affect the policy.

For example, using the CSP-Global-Gadget-Pref header as an example, if you want the value for `AT` to be evaluated to determine if the policy should apply, specify the name of the header parameter as:

```
CSP-Global-Gadget-Pref:AT
```

If you want the entire string to be evaluated without assigning meaning to name= value pairs, you simply specify the name of the header parameter as:

```
CSP-Global-Gadget-Pref
```

You can create a rule that matches when the identified header is provided with an empty value or when it is absent from the request. In this example, `Standards:release` would be considered empty, and `Standards:SOAP-ENV` would be considered absent because it is ignored:

```
Standards: type=SOAP;SOAP-ENV:mustUnderstand="1",release=,version=1.2
```

Client IP

When a policy rule uses the Client IP parameter, the WebAccelerator looks at the IP address of the client making the request to an WebAccelerator. The client could be a browser or it could be a remote WebAccelerator, such as the F5 WebAccelerator Remote. The IP address is not always the address of the client who originated the request.

Chapter 5

The Request Type Hierarchy

Managing the Hierarchy ◂
Hierarchy Inheritance ◂
Request Type Hierarchy Example ◂

All of the policies that you define for the WebAccelerator are organized and managed through the Request Type Hierarchy. Existing policies are accessed in the Admin Tool by selecting their node in the Request Type Hierarchy. Once you select the node, you can see the policies associated with it and you can edit or add to those policies. To create a new group of policies, you create a leaf node in the Request Type Hierarchy, and create a matching policy for the node. This matching policy defines what types of requests and responses match the node, and in turn, defines the node. You create caching policies for the node, and these caching policies are applied to any request or response that matches the node. Deleting a node deletes the policies associated with it.

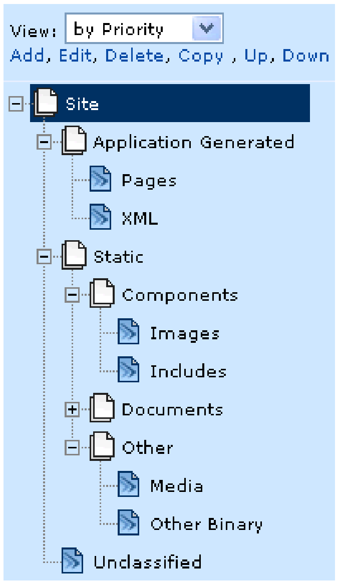
The tree structure of the Request Type Hierarchy enables parent-child relationships, so that a policy associated with a leaf node inherits rules from the policies of its parent nodes. A request or response can only be matched to a leaf node in the hierarchy, and assigned to that node. Parent or ancestor nodes exist only for the ability to inherit their policies.

Leaf nodes can also be assigned priorities, so that if a request matches two leaf nodes, the node with the higher priority takes precedence.

Managing the Hierarchy

You create and manage the Request Type Hierarchy using the Policy Editor. To access the Request Type Hierarchy, log into the Admin Tool and go to Policies. Click the Edit link for the policy set you want to change. The hierarchy is shown at the left of the screen, and you can choose to view it in Alphabetical or Priority order:

Figure 9 A View of a Request Type Hierarchy



While in the Policy Editor, you can change the Request Type Hierarchy by:

- adding a node (Add)
- editing or renaming a node (Edit)
- deleting a node (Delete)
- copying a node (Copy)
- changing a node’s priority (Up, Down)

For specific details on managing the Request Type Hierarchy, see the online help.

Node Priority

A node’s priority affects application matching. When application matching matches a request to two nodes equally, the request is matched to the node with the highest priority and that node’s caching policies are applied.

To change a node’s priority, switch to the Priority view. Up and down arrows appear. Select the node, and click either the up or down arrow to change its priority. You can only change priority within a branch of the tree. Using Figure 9 on page 56 as an example, you could give `Default` priority over `Search`, but not over `Images`.

Hierarchy Inheritance

The WebAccelerator performs matching only against the leaf nodes in the Request Type Hierarchy. Requests that match a leaf node then have the other policies on the node applied to them. Policies defined for nodes that are not leaf nodes are only used for inheritance, never applied directly to a request or response. In the Request Type Hierarchy, you create nodes that are not leaf nodes because you want to define certain policies that a group of leaf nodes can share.

For example, if you want to create several leaf nodes with similar matching policies that use a certain variation policy, you can create them under the same parent node, and define the variation policy at the parent. The leaf nodes inherit the policy from the parent. This technique enables you to create policies faster and more consistently than recreating a policy over and over again for different leaf nodes.

Each policy consists of rules based on HTTP Request Data Type Parameters. A policy can be a combination of locally created rules and inherited rules. For example, a parent node could have a proxying policy that consists of a rule based on a Path Segment parameter. At one of its child nodes, you could create a proxying rule based on the Cookie parameter. For that node, the proxying policy now consists of two rules, one inherited and one not.

Origin of Rules in a Policy

Any rule in a policy can be created by:

- local definition, in which the rule was created at the current node and not inherited from any ancestor

  For this to be the case, there must not be a version of the rule defined for any of the node’s ancestors. You cannot define a rule at a node and turn off inheritance so that children of the node do not inherit the rule. All rules are inherited unless the rule is defined at a leaf node (in which case there are no children to inherit it).

- inheritance, in which all settings and options for the rule are inherited from an ancestor node

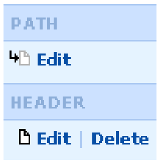
  This is the case whenever a policy is defined for a parent node, and the child nodes do not override it. To edit this policy, you must find the ancestor node where it was defined and edit it there. The changes you make affect all child nodes that inherit this policy.

- a combination of inheritance and overrides, in which a policy is inherited but some or all of the rules in the policy are overridden at the child nodes

  At any child node, you can choose to edit an inherited policy. This does not change the actual policy, which must be edited at the node where it was defined. However, this does change the copy of the policy in effect at this node, overriding any inherited rules that you edit with the changes you make. Any nodes that are children of this node inherit the changed (overridden) policy, and can also override rules in the copy they inherited. See "Overriding Inherited Policies" on page 59.

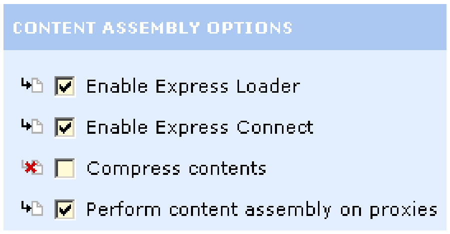
Inheritance Indicators

When you view a policy, there are icons that indicate the inheritance status of the policy. In the figure below, this policy has two rules, and the icon under Path indicates its rule is inherited while the Header parameter rule is not inherited:



The Header rule can be deleted by clicking the Delete link, but the Path rule cannot because it was not created at this node. You cannot delete a rule that is inherited. You can only modify values or options set for the rule. If you click on the icon under Path, a pop-up window appears that tells you the ancestor node where the Path rule was defined.

This rule for an assembly policy has several options, and some of these options are inherited as is from the parent, while one is inherited but overridden:



☑ Enable Express Loader
☑ Enable Express Connect
☐ Compress contents
☑ Perform content assembly on proxies

The icon with the red slashed circle indicates the option was inherited but overridden. At the parent node, the other settings must have been unchecked, but `Compress` `contents` was checked, and at this child node, it must have been unchecked, to override it.

Note: The override icon only shows overrides occurring at the current node. You cannot see if a parent node inherited the policy and overrode a setting that this node inherited. You must click on the parent node to see its policies and whether it inherited and overrode a rule or option.

Overriding Inherited Policies

Policies are usually defined by one or more rules based on parameters (see "Specifying Parameters" on page 47). When a policy is inherited from a parent node, the child node inherits all the rules defined for the policy. You can add rules to the policy but you cannot delete or remove any of the inherited rules at the child node. For example, if a node inherits a matching policy with a rule based on the Extension parameter and another rule based on a Cookie parameter named `SessionID`, you cannot delete either rule and you cannot change the Cookie rule to be based on a cookie of a different name. In other words, you cannot change the identity of the parameter the rule is based on.

When you click on the Edit link for an inherited rule, you are taken to an editing screen where you can change the value set for the parameter and any options associated with it, such as WebAccelerator behavior. For example, if you edit the rule for `SessionID`, you can change the value specified for `SessionID` and whether a request should match the value or not match the value, or specify whether the request should match if `SessionID` is absent from the request. The changes you make override the inherited settings for the parameter.

Once you have overridden a rule, it is always possible to revert to the original inherited definition by clicking on the Revert link next to the parameter rule you want to revert.

At any level in the Request Type Hierarchy, you can override settings in a rule and these overrides are passed along to all the node's children, in addition to any new rules defined for the node. In a deep Request Type Hierarchy, where leaf nodes have many ancestors, it can be complicated to trace where the settings for each rule came from. If you click on either the inherited or override icon, a pop-up box appears that shows you the node where the rule was first created, and you can go to that node, to see the rule's original settings.

Request Type Hierarchy Example

Suppose your site receives four basic types of requests:

- requests for your home page:
  `http://www.somesite.com/`
  `http://www.somesite.com/index.jsp`
- normal requests for content at your site:
  `http://www.somesite.com/apps/doSomething.jsp`
  `http://www.somesite.com/apps/doSomethingElse.jsp`
  `http://www.somesite.com/apps/doAnotherThing.jsp`
- requests for search applications at your site:
  `http://www.somesite.com/srch/doSearch.jsp`
  `http://www.somesite.com/srch/doSimpleSearch.jsp`
- requests for graphics images:
  `http://www.somesite.com/images/<`*`someimage`*`>.gif`

Tasks

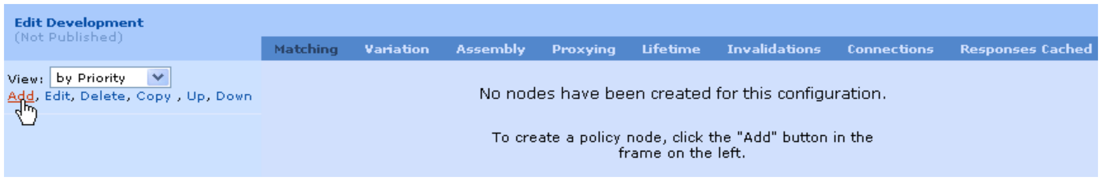
Create three top-level nodes in the Request Type Hierarchy. The names of the nodes are arbitrary but you might call them:

- Home, defines the caching policies related to your home page
- Applications, defines the caching policies related to the applications at your site. This node might have two child nodes:
  - Default, defines the caching policies related to non-search related applications
  - Search, defines the policies related to your site's search application
- Images, defines the policies related to graphics images

Actions

To create a top-level node in your Request Type Hierarchy:

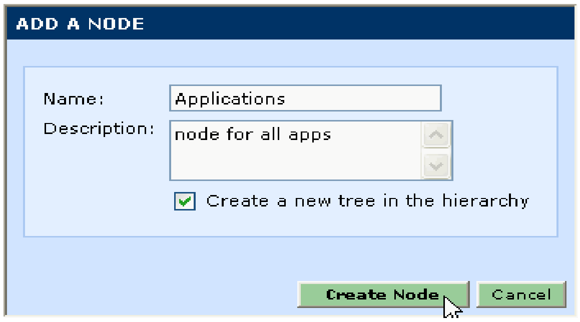
1. While in edit mode (you can see Edit Development is highlighted), click the Add button in the Request Type Hierarchy:



No nodes have been created for this configuration.

To create a policy node, click the "Add" button in the frame on the left.

2. In the `ADD A NODE` pop-up window, enter the name of the node and, optionally, a description for the node. Also, make sure to check `Create a new tree....` This ensures the new node starts a new tree. Once you have done that, click the Create Node button:



Name: Applications

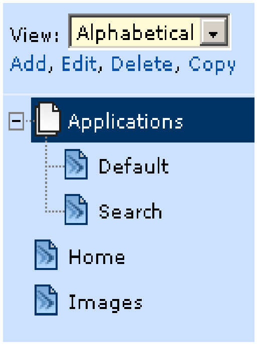
Description: node for all apps

[✓] Create a new tree in the hierarchy

To create the child or leaf nodes under the `Applications` node, repeat the steps except:

1. Select the Application node before you click the Add button.
2. Do not check `Create a new tree....`

When you are done, your hierarchy should look like:



You then have to define both application matching parameters and caching policies defined for each of the nodes in this hierarchy.

To define matching policies, see Chapter 6, Setting Matching Policies. For other types of policies that you can define for each node, see:

- Chapter 7, Setting Variation Policies
- Chapter 8, Setting Assembly Policies
- Chapter 10, Setting Proxying Policies
- Chapter 11, Cache Control and Content Lifetime
- Chapter 12, Invalidating Cached Content
- Chapter 13, Connection Mapping
- Chapter 14, Setting Responses Cached Policies

Chapter 6

Setting Matching Policies

Supported HTTP Request Data Types
Defining Matching Policies
Application Matching and the Request Type Hierarchy
Application Matching Example

Application matching is the process that the WebAccelerator uses to map an HTTP request or response to a set of caching policies. Application matching compares the information the WebAccelerator observes in the HTTP request or response with the rules set by matching policies.

Each node in the Request Type Hierarchy has a set of matching policies and a set of caching policies associated with it. The matching policies are used by the WebAccelerator to determine if an HTTP request or response matches that node. The caching policies for that node are applied if there is a match to the node.

For example, you might have a node in your Request Type Hierarchy specifically for image files. For this node, you define a matching policy based on the extension found on the HTTP request, and requiring common image file extensions. When a request contains an extension of `gif`, `GIF`, `jpg`, `JPG`, `jpeg`, or `JPEG`, the request is matched to the image node, and its caching policies, which you defined to handle image files appropriately for your site, are applied.

Supported HTTP Request Data Types

The HTTP request data types on which you can base matching policies are:

- Protocol
- Host
- Path
- Extension
- Method
- Query Parameter
- Unnamed Query Parameter
- Path Segment
- Cookie
- User Agent
- Referrer
- Header
- Client IP

Supported HTTP Response Data Types

The HTTP response data types on which you can base matching policies are:

- Content-Disposition
- Content-Type

Unlike the request data types, you do not base a matching policy directly on the value of one of these response data types. Instead, the values are used to classify a response and generate a content type, and you can base a matching rule on this content type.

Before application matching occurs, the request or response is classified by object type. For requests, the classification is based on the file extension present in the request. For responses, the classification is based on the first information present, in this order:

1. the file extension from the filename field in the Content-Disposition header in the response
2. the file extension from the extension field in the Content-Disposition header in the response
3. the Content-Type header in the response, unless it is an ambiguous MIME type

4. the extension of the path in the request

Once the request or response is classified by object type, a content type is generated for it by appending the object type to the group in which the object type belongs: *`group.objectType`*

The groups and object types are defined in the global configuration fragments file. For more information, see "Classifying Responses" on page 213.

You can specify a matching rule based on the Content Type parameter by specifying the regular expression you want a request or response's content type to match or not match. For a list of the possible content types a request or response might have, and more information on using this parameter, see the online help.

Defining Matching Policies

You manage matching policies using the Policy Editor.

1. Log into the Admin Tool.
2. Go to the Policies screen.
3. Click the Edit link for the policy set whose matching policies you want to change or create.
4. Select the Matching link.

Matching Parameters

Matching policies consist of rules based on the HTTP Request Data Type Parameters and the Content Type parameter. You identify the parameters to look for in a request, and the values to attempt to match against. Each node in the Request Type Hierarchy has its own matching policy with a unique set of rules. The WebAccelerator attempts to match the request to one node, trying to find the best match as described in "Application Matching and the Request Type Hierarchy".

In order for an HTTP request to match to the parameter as defined by the rule, the parameter must appear on the request in the state defined by the rule. For example, you can create a matching rule based on a Cookie parameter named `sessionID`. You choose in what state the `sessionID` cookie needs to appear in the request. Your rule could specify one of these states:

- `sessionID` is absent; it does not appear on the request, or
- `sessionID` is empty; it appears but has no value set for it, or
- `sessionID` appears and its value must match some value you specify, for example, its value must begin with a number

If the state you specify is true, the request is a match for this rule. If the request matches all the other rules in the matching policy for this node, it is considered a match for the node and assigned to the node. All other policies set for this node, such as variation and lifetime, are applied to the request.

The matching for most parameters is either case-insensitive or controlled by a checkbox that allows you to specify whether you want it to be case-sensitive or insensitive.

Application Matching and the Request Type Hierarchy

Matching policies are organized using the Request Type Hierarchy, described in Chapter 5, The Request Type Hierarchy.

Application matching is only performed against leaf nodes in the hierarchy. Each leaf node must have its own matching policy, with one or more parameter rules defined for it that are inherited or created locally for the node. For information on inheritance support in the Request Type Hierarchy, see "Hierarchy Inheritance" on page 57.

In order for an HTTP request or response to match to a specific leaf node, it must match all the matching rules defined for the leaf node. There are precedence rules for this matching process, but they only apply if a request or response matches all the rules for two or more nodes.

Application Matching Precedence Rules

The basic principles for application matching are simple:

- the most specific match possible is chosen over a more general match
- a superset is chosen over a subset

For example, there are two nodes, one with matching rules based on three query parameters and the other with rules based on two of the same three query parameters. A request contains a match for all three query parameters, so both nodes match but the node with three rules is chosen as the best match for the request because it is the superset. If two nodes both have matching rules based on the Path parameter, and a request matches both, the node with the rule specifying the longest (most specific) path is chosen.

Now consider the case where a node has three rules, based on query parameters, and another node has two rules based on two completely different query parameters, and a request matches all the rules. No node is chosen because the rules are not supersets or subsets of each other, so the matching protocols cannot say that a complete match to one node is a better match than a complete match to another node.

If the precedence rules cannot choose one node over another, the result is considered ambiguous so node priority is used to choose a node. You can assign a priority to a node when you create it, and you can modify it later. See "Node Priority" on page 56.

In addition to these basic principles:

1. Matching based on the Path parameter is always chosen over other matches, and an exact path match is always chosen over other path matches. An exact path match is one where the value set for the Path parameter includes the `?`

   For example, if you have a rule with this value set for Path:
   `apps/srch.jsp?`

   and you receive this request:
   `http://www.swanlabs.com/apps/srch.jsp?value=computers`

   it is considered an exact match to the rule. See "Paths As Prefixes" on page 68.

2. If two nodes use matching rules based on a path and a request's path matches both rules, then the request is matched to the node whose rule specifies the longest path. For example, if one rule specifies a path of:
   `/apps`

   and a second specifies a path of:
   `/apps/search`

   then this request is matched to the node of the second rule:
   `http://www.somesite.com/apps/search/simpleSearch.jsp?....`

3. Matching rules based on Extension have the next highest priority. If the extension on a request matches the extension specified by an application matching rule, then the request is matched to the node to which the rule belongs.

Unmatched Requests

If a request does not match any leaf nodes in the Request Type Hierarchy, the request is proxied to your origin servers. Remember, a request must match all the matching rules to be considered a match. If a request matches all the rules for a node except one, it is not a match and is proxied. Performance Reports compiles statistics on this unmatched traffic so you can analyse it.

There are a set of predefined policies used to handle unmatched requests and responses. If a request or response is not matched to a node, this special set of policies is used to determine how to handle it. If a request is matched, but the response is not matched, the portion of the unmatched policy set that is relevant to responses is used to handle the response.

Application Matching Example

Suppose your site receives the types of requests described in "Request Type Hierarchy Example" on page 60. As a result, you have created this Request Type Hierarchy:

```
-Applications
  -Default
  -Search
-Home
-Images
```

Tasks

Create these matching rules for each node in the hierarchy:

Node	Application Matching Parameter
Home	Create a parameter based on the Path data type. Provide two values for the Path parameter: `/index.jsp` `/?`
Default	Create a rule based on the Path data type. Provide one value for the Path parameter: `/apps`
Search	Create a rule based on the Path data type. Provide one value for the Path parameter: `/srch`
Images	Create a rule based on the Path data type. Provide one value for the Path parameter: `/images`

Paths As Prefixes

A path of / and /? are two different things. By default, a path that you provide for a policy is a prefix. For example, if you give the parameter a path of:

```
/a/b
```

both these requests match:

```
http://www.bowwow.com/a/b?treat=bone
http://www.bowwow.com/a/b/c?toy=ball
```

However, a question mark (?) indicates that an exact match must be made. If you specify this value for a Path parameter:

```
/a/b?
```

only this request matches:
`http://www.bowwow.com/a/b?treat=bone`

Actions

For each of the nodes identified in the preceding section, create your matching policies as follows:

1. Select the node for which you want to create the policy.
2. Select the Matching link to be sure you are creating a matching policy.
3. Select Path from the Add Parameter drop-down list.
4. Click the Add button.
5. In the Path parameter screen, enter the values on which you want to base application matching. For example:
   ```
   /index.jsp
   /?
   ```
6. When you are done, click the Save button.

You now see the parameter in the Matching summary table for the node.

In order for the new policy to be in effect for your site, it must be published. See "Publishing Policies" on page 45 for more information.

Chapter 7

Setting Variation Policies

Content Variation and Caching ◀
Default Caching Behavior ◀
When Variation Policies Are Required ◀
When Variation Policies Are Optional ◀
Defining Variation Policies ◀
Content Variation and the Request Type Hierarchy ◀
Content Variation Example ◀

Variation policies indicate the elements on a request URL that affect the content returned by your web servers. This information is used by the WebAccelerator to create a UCI (Unique Content Identifier) for the request and for the cached page, stored in the form of a compiled response, that provides the response. The compiled response is stored in cache under this UCI. HTTP request elements that do not affect page content are ignored and are not used in the UCI. As a result, the values set for those elements are not used to identify unique instances of cached content.

The variation policies are applied after the request is matched to a node, and the UCI is created at that time. It does not matter what node a response matches to. Only the variation policies for the request's node are used.

For example, by default the WebAccelerator assumes that the presence or absence of a cookie and the cookie's value do not change the contents of a web page. However, sometimes the value set for a cookie changes the content served by your site. Sometimes the lack of a specific cookie on a request means that your site serves a different page than when the cookie is set. In these cases, you can set a variation policy to specify that the cookie presence or value is significant for content. The cookie and its value are used in the UCI for any request assigned to this node. In order for the UCI of a new request and the UCI of a cached page (stored in the form of a compiled response) to match, the new request must have the same value for the cookie as the request that generated the cached page. If there is a match, that page is used to service this new request.

Variation policies apply even when a request is proxied because they affect the UCI under which the response is cached. Variation policies do not apply to requests that are always proxied and never served from cache.

Content Variation and Caching

The fundamental principle of content variation is that you can determine what content should be in a response based on the elements you see in the request. Given this, you can look at a request and decide which elements in it affect the content served, and which elements should be ignored because they do not affect the content. You could extract from the request only those elements that affect content, and use them to create an identifier that represents both the request and the page generated by the request. This is the UCI created by the WebAccelerator. All cached pages are stored under the UCI of the request that generated the page. The UCIs of new requests are matched against the UCIs of cached pages to find an appropriate page to serve.

For example, perhaps you have 12 slightly different requests, but they only generate 5 different responses. Ideally, the WebAccelerator would ignore the appropriate elements in the requests, so that all the requests that generate one particular response would receive the same UCI. The first request results in a proxy, because there is no matching UCI in cache, and the response is cached under the request's UCI. The next request for the same page is assigned the same UCI, and that UCI is used to find the cached page, and so on. Every request with a previously unseen UCI results in a proxy for a new page, which is then cached. After handling all 12 requests, there should be 5 pages in cache, each with a unique UCI. This is the correct handling of the requests and an efficient use of cache.

If the WebAccelerator ignores too many elements in the requests, it would create UCIs based on too small a subset of elements. The 12 requests might result in only 3 unique UCIs, and be served only 3 pages, with some requests incorrectly getting identical content. If the WebAccelerator does not ignore enough elements, the UCI for each request might be unique, since the requests are all slightly different. The 12 requests might result in 12 different UCIs, with each request being proxied and its response cached under a different UCI. Although each request receives the correct response, the cache now has 12 pages, some with the same content, instead of 5 unique pages. If this happens for every request a WebAccelerator receives, you end up with a large cache that is never used to service a request.

The WebAccelerator has certain default assumptions about which elements to ignore or include when creating a UCI. The variation policies change these assumptions and also refine them. For example, assume two requests are received by the WebAccelerator, and are identical except for the value of a particular query parameter called `Version`. By default a different UCI is created for each, and both responses are cached, each under its unique UCI.

If you create a variation policy that specifies that all values of `Version` define the same content, the WebAccelerator no longer includes the value of `Version` in the UCI. Both requests would have the same UCI and could be served the same page.

Default Caching Behavior

You do not have to set a variation policy for every element expected on your HTTP requests. Instead, set a variation policy only for those elements that vary from the default. The WebAccelerator assumes these elements are significant for content:

- Host
- Query Parameter
- Unnamed Query Parameter
- Path Segment
- Protocol

Create a variation policy with rules for these parameters if these elements are not significant for content. In other words, if their presence and value in a request can vary while the response remains the same, create a rule to describe that to the WebAccelerator. When the WebAccelerator assembles a UCI, the elements are not included.

The WebAccelerator assumes these elements are not significant for content:

- Method
- Cookie
- User Agent
- Referrer
- Header
- Client IP

Create a variation policy with rules for these parameters if these elements are significant for content. In other words, if their presence and value in a request results in different responses, create a rule to describe that to the WebAccelerator. The WebAccelerator includes their values in the UCI.

When Variation Policies Are Required

You must create a variation policy whenever a request element the WebAccelerator assumes does not affect content actually does affect the content served by your site. That is, you must create a variation policy whenever:

- the method used for a request can change the page that your site serves
- the state of a cookie can change the page that your site serves
- the connecting client’s IP address can change the page that your site serves

- the state of the HTTP USER_AGENT, REFERER, or other request headers can change the page that your site serves

If you do not create a variation policy in these situations, the WebAccelerator can serve the wrong content when servicing a request. The WebAccelerator ignores the element and excludes it from the UCI. Requests where that element has different values can all have the same UCI and therefore match to the same compiled response, getting the same response. If different values of the element should generate different pages, these requests are getting an incorrect response.

For example, suppose your site has an application that can produce two different pages, depending on the value set for the `version` cookie. If `version` is set to a value of 1, your site produces customer version of a page. If `version` is set to a value of 2, your site produces a partner version of the page.

If you do not create a variation policy that specifies the `version` cookie as significant for content, the WebAccelerator produces at most two compiled responses, one for requests with the cookie and one for requests without. The value set for the cookie is ignored. When the first request that contains the cookie is received, a UCI is generated, the request is proxied, and the response is cached under the UCI. If the request contained `version=2`, the page that is cached under the UCI is for partners. All subsequent requests that include the cookie end up with the same UCI and so are served the cached partner page, even when `version =1` and they should be served the customer page.

To solve this, create a variation policy that identifies the `version` cookie as significant for content. This causes the WebAccelerator to include the value of `version` when it creates a UCI and the cache reflects this by containing three compiled responses:

- one for content produced for `Cookie: version=1`
- one for content produced for `Cookie: version=2`
- one for content produced when the `version` cookie does not appear in the request

When Variation Policies Are Optional

There are times when creating a variation policy does not affect the accuracy of the pages served, but does affect cache size. For request elements that the WebAccelerator assumes are significant for content, but which your site actually ignores in determining content, you can save cache space by setting a variation policy.

Request elements that are assumed to be significant for content are:

- Host
- Query Parameter

- Unnamed Query Parameter
- Path Segment
- Protocol

For example, suppose your site uses a query parameter that provides session tracking information. Your site's pages are not affected by the value set for the session tracking query parameter. You can use a variation policy to identify the session tracking query parameter as not significant for content. This allows the WebAccelerator to cache one version of the page to be used for any value of the session tracking query parameter, instead of caching separate but identical pages for each different value.

The extent of the resource savings could be large. Without the appropriate policy, the WebAccelerator creates one unique compiled response for every page that every user views. This results in an unnecessarily large cache. With the appropriate policy, however, the size of your cache is reduced and your cache efficiency is greatly enhanced.

Defining Variation Policies

You manage variation policies using the Policy Editor:

1. Log into the Admin Tool.
2. Go to the Policies screen.
3. Click the Edit link for the policy set whose variation policies you want to change.
4. Select the Variation link.

For details on the screens you use to define variation policies, see the online help.

Variation Parameters

Variation policies consist of rules based on the HTTP Request Data Type Parameters. These rules determine what elements in a request are used in the UCI. Each node in the Request Type Hierarchy has its own variation policy. Once a request is matched to a particular node during application matching, the rules for the variation policy for the node are applied to the request. If the request contains parameters with values that match the rules, the behavior specified in the rules is applied to the request. You can specify one of two behaviors:

- HTTP request elements that match this rule result in unique page content

  The WebAccelerator uses the specified parameter and its value in the UCI created for this request.

- requests that match this policy do not result in unique page content

  The WebAccelerator ignores the value set for the identified parameter, which means that the parameter and its value are not included in the UCI created for this request.

After the UCI is created, the WebAccelerator can use it to find a compiled response in cache that has the same UCI. If there is a matching compiled response, it can be used to service the request. If there is not a matching compiled response, the request is proxied to the origin servers. The response is then turned into a compiled response and cached under the UCI before being served to the requesting client.

Value Groups

Instead of specifying a single value rule for a variation parameter, you specify a value group. A value group is a collection of value rules. The main purpose of a value group is to enable you to specify several rules for the same parameter, each with its own set of values indicating a certain behavior.

For example, a matching policy based on cookie A can only specify a single set of values for the cookie and whether that is a match or not a match. A variation policy can specify a value group for cookie A, consisting of several rules such as:

- when cookie A begins with `aa`, it indicates page content is the same
- when it begins with `bb`, page content is the same
- for all other values, page content is unique

As you can see, you can specify different behaviors for different values, instead of a single behavior. This is the flexibility a value group allows.

In this case, these rules tell the WebAccelerator that for all requests containing a cookie called A whose value begins with `aa`, those requests should all map to the same page. If the value began with `bb`, those requests also map to one page, but a different page than for `aa`. For requests with a cookie called A whose value begins with anything other than `aa` or `bb`, each request would map to a separate unique page, one for each value seen for the cookie. That is, if there were three requests with three different values for cookie A, such as `ca233`, `ca234`, `ba234`, they would map to three different pages. Whereas a request with cookie A equal to `bb323` and another request with cookie A equal to `bb422` map to the same page.

If you specified these rules differently, for example as:

- when cookie A begins with `aa` or `bb`, it indicates page content is the same
- for all other values, page content is unique

the behavior would be different. In this case, requests with cookie A beginning with `aa` or `bb` all map to the same page, instead of `aa` requests mapping to one page and `bb` requests mapping to another page.

Multiple Matching Parameters

If an element in a request matches two or more variation parameters, and the parameters define conflicting behavior (one parameter indicates that the value is significant for content and the other does not), the parameter is considered to be significant for content, resulting in a separate compiled response for every value of the parameter actually seen on a request. This is the most reliable way to handle the conflict.

Ambiguous Query Parameters

There are situations in which the WebAccelerator cannot tell what variation parameter to use. This occurs when you set conflicting parameters for named and unnamed query parameters by doing something like:

Name	Value Match	Affect on Content
All other query parameters	All values	unique content
Unnamed in ordinal 1	All values	same content

Given this policy, the WebAccelerator does not know how to handle this request without some guidance from you:
`http://www.somesite.com/show.jsp?computer&vendor=whiteBox`

This request can be interpreted to mean that `computer` is either:

- a query parameter named `computer` that has no value set for it, or
- an unnamed query parameter in ordinal 1 whose value is set to `computer`.

Depending on how you interpret this, `computer` should be used in the UCI or not, and result in a unique page or not. By default the WebAccelerator treats all ambiguous query parameters as a named query parameter without a value. You can override this default behavior by editing the `All other query parameters` variation parameter. In that screen, there is a control that tells the WebAccelerator to treat ambiguous query parameters as unnamed.

Content Variation and the Request Type Hierarchy

Variation policies are organized using the Request Type Hierarchy. The Request Type Hierarchy is described in Chapter 5, The Request Type Hierarchy.

Variation policies are applied only against leaf nodes in the hierarchy. Each leaf node in the Request Type Hierarchy has either one or no variation policy defined for it. A variation policy must have at least one variation parameter rule.

The set of variation parameters defined for a leaf node are a combination of the parameter rules defined locally at the node and parameter rules inherited from the node's ancestors. For information on inheritance support in the Request Type Hierarchy, see "Hierarchy Inheritance" on page 57.

Variation policies specify which HTTP request elements to base the WebAccelerator caching behavior on:

1. The WebAccelerator maps an HTTP request to a leaf node in the Request Type Hierarchy. See Chapter 6, Setting Matching Policies.
2. Once mapped, the WebAccelerator decides if it should attempt to service the request from cache or if it must proxy the request to your origin servers. See "Servicing HTTP Client Requests from Cache" on page 192.
3. If the WebAccelerator decides it can attempt to service the request from cache, it examines the leaf node's policies to see if a variation policy is defined for the node.
4. If a variation policy is defined, the WebAccelerator examines each of the policy's parameter rules to see which elements of the HTTP request it should use for UCI creation. Each parameter rule overrides the WebAccelerator's default assumptions about which elements affect page content.
5. The WebAccelerator generates a UCI based on the elements that are defined as affecting page content.
6. The WebAccelerator uses this UCI to find a valid compiled response in cache. If it finds a compiled response cached under that UCI which has not expired, the WebAccelerator uses the compiled response to service the request.
7. If a valid compiled response is not available, the WebAccelerator proxies the request to your origin servers for handling.
8. Upon receiving a response from the origin servers, the WebAccelerator creates a compiled response and caches it under the UCI it created in Step 5.
9. The WebAccelerator uses the compiled response to service the request.

Content Variation Example

Suppose your site receives the types of requests described in "Request Type Hierarchy Example" on page 60 and your Request Type Hierarchy looks like:

```
-Applications
  -Default
  -Search
-Home
-Images
```

Further, suppose that your site changes branding information for any application that it uses if the REFERER request header begins with:

```
http://www.myportalpartner.com
```

but that all other values for the REFERER request header are not meaningful for content.

Also, suppose your site uses a query parameter called `sessionID` to track the users of your site. This query parameter does not affect page content, and is used for tracking purposes only.

Tasks

In this situation, create two variation policies. Place one policy on the Applications node (see "Request Type Hierarchy Example" on page 60) that describes the conditions under which the REFERER request header is meaningful for content. Place the other policy on the Search node to identify `sessionID` as not meaningful for content.

Actions

Step 1 Create REFERER Policy

To create the variation policy based on the REFERER data type:

1. Access the Policy Editor and select the Variation link.
2. Select the Applications node in the Request Type Hierarchy.
3. Find the REFERRER section of the Variation summary table and click the Edit link.
4. In the `Value Groups` section of the Edit Parameter screen, click the Add button.

5. Check the Value MATCHES box and enter the regular expression that matches the value you expect on the REFERER request header, which is:
   `http://www\.myportalpartner\.com.*`

   For more information on regular expressions, see “Regular Expressions” on page 221.
6. Indicate that matching requests result in unique page contents by choosing `Different Content` from the drop-down list.
7. Click the OK button.
8. On the Referrer page, click the Save button.

You now see the new parameter in the Variation summary table.

You must publish your policies in order for the WebAccelerator to use this parameter. For more information on publishing policies, see “Publishing Policies” on page 45.

Step 2 Create Query Parameter Policy

To create the variation policy based on the Query Parameter data type as is suggested in “Tasks” on page 79:

1. Select the Search node in the Request Type Hierarchy.
2. Select `Query Parameter` from the Add Parameter drop-down list.
3. Click the Add button.
4. In the Add Parameter screen, enter the query parameter’s name (`sessionID`) in the `Name` field.
5. Click the Add button in the Value Groups table.
6. Select the `Value MATCHES` check box and enter a regular expression of:
   `.*`

   This means that the query parameter can have any value.
7. Also check `Value is an EMPTY String`.
8. Make sure that the Value Group is set to indicate that page content is not affected by this parameter by selecting `Same Content` from the drop-down list.
9. Click the OK button.
10. Click the Save button.

The new parameter now appears in the Variation summary table for the Search node.

You must publish your policies in order for the WebAccelerator to use this parameter. For more information on publishing policies, see “Publishing Policies” on page 45.

Chapter 8

Setting Assembly Policies

Assembly Policies
Parameter Value Substitution
Express Loader Management
Express Connect Management
Compressing Content
Content Assembly on Proxies
Rewrite Engine Scripts
Defining Assembly Policies
Content Assembly and the Request Type Hierarchy
Content Assembly Example
Using the Debugger

The WebAccelerator manages content by compiling it into a response. Compiled responses are executable applets that the WebAccelerator uses to assemble a page needed to service an HTTP request.

The WebAccelerator services HTTP requests by running a compiled response using whatever input parameters are required to reassemble content. There are two forms of input parameters that the WebAccelerator can use when it runs a compiled response. The first are your assembly policies, which identify:

- how to change parameter values found on embedded URLs by substituting values from the request or randomly generated values
- various content assembly options, such as compression

The second are the page assembly instructions. You can use ESI markup tags to specify HTML fragments to include in the page. This is described in Chapter 9, Assembling Content with ESI.

Assembly Policies

You use assembly policies to describe:

- parameter value substitution requirements. Parameter value substitution is described in the next section. A special case of parameter value substitution is described in "Random Value Substitution" on page 84.

  Parameter value substitution (including random value substitution) rules are ignored in cases where they are in conflict with ESI markup. See "Serving ESI Content from the Cache" on page 99 for more information. Also, parameter value substitution is only performed for pages that the WebAccelerator serves from cache – it does not perform substitution when it has just proxied for fresh content and is serving that response back to the requesting client.

- content assembly options, such as:
  - whether Express Loader should be used (see "Express Loader Management" on page 84)
  - whether Express Connect should be used (see "Express Connect Management" on page 86)
  - whether content served by the WebAccelerator should be compressed using gzip encoding (see "Compressing Content" on page 88)
  - whether assembly should be performed on proxies (see "Content Assembly on Proxies" on page 89)

Content assembly is performed after the response is matched to a node, so this node's assembly policy is used. If the response matches to a different node than the request, the assembly policy on the request's node is irrelevant. This is true even if the request is not proxied. When the response was originally cached as a compiled response, the assembly instructions from the response node's assembly policy were stored as part of the compiled response. When a response is served from cache, it is these stored instructions that are used.

In addition to the assembly mechanism provided by the WebAccelerator, you can also use scripts to process and change responses received from your origin servers. This is performed using the Rewrite Engine, which is a mechanism that provides an efficient scripting language intended for the manipulation of HTTP responses. See Appendix B, Rewrite Engine, for more information.

Parameter Value Substitution

Parameter value substitution changes the value for a targeted parameter when a page is served from cache. The targeted parameter appears on a URL embedded in a web page.

This is common when a query parameter is used to contain identification information for the site's visitors. Pages served in response to these requests often have hyperlinks included on them that require that information to appear on the pages. By using assembly policies to identify these situations, the WebAccelerator can take the value presented for an identified source on an HTTP request, and place that value on the appropriate locations in URLs embedded in a page that it is serving.

Without parameter value substitution, whatever value (derived from the original request) was cached is the value used when servicing all subsequent requests, even though the requests use different values which should be used in the response. For example, if the original request URL that generated the cached page was:

```
http://www.somesite.com/apps/shopping.jsp?shopper=AAnb35vnM09&....
```

the embedded URL in the cached page might be:

```
<a href="http://www.somesite.com/apps/shoppingCart.jsp?shopper=
AAnb35vnM09&....">link</a>
```

A subsequent request for the page:

```
http://www.somesite.com/apps/shopping.jsp?shopper=SNkj90qcL47&....
```

is still served the cached page which contains:

```
<a href="http://www.somesite.com/apps/shoppingCart.jsp?shopper=
AAnb35vnM09&....">link</a>
```

If you create an appropriate parameter pass-through policy, the WebAccelerator substitutes the original cached value for `shopper`, replacing it with the value on the request:

```
<a href="http://www.somesite.com/apps/shoppingCart.jsp?shopper=
SNkj90qcL47&....">link</a>
```

Note: If the source you specify for the substitution does not appear on the request, the parameter is removed entirely from the qualifying embedded URLs. In the above example, if a subsequent request was simply:

```
http://www.somesite.com/apps/shopping.jsp
```

then embedded URLs served to the requestor are:

```
<a href="http://www.somesite.com/apps/shoppingCart.jsp?">link</a>
```

Random Value Substitution

Random value substitution is similar. Random value substitution generates a random number and places that number on a targeted location in an embedded URL. When the WebAccelerator compiles the page into a response, it examines the target location to see the length of the string used for the value. On subsequent page requests, the WebAccelerator replaces that value with a random number of the same length.

This feature is generally used for sites that make use of an internet advertisement agency. They require random numbers to be placed on URLs that request ads as a way to break traditional caches. By requiring a random number, these internet advertisement agencies force all such traffic to their site.

For example, suppose a cached page includes the embedded URL:

```
<iframe src="http://www.aaaa-Ads.com/getAd?entity=45&...”></iframe>
```

With random value substitution, subsequent requests for the page result in random values set for the `entity` query parameter:

```
<iframe src="http://www.aaaa-Ads.com/getAd?entity=45&...”></iframe>
<iframe src="http://www.aaaa-Ads.com/getAd?entity=11&...”></iframe>
<iframe src="http://www.aaaa-Ads.com/getAd?entity=87&...”></iframe>
```

Express Loader Management

Express Loader is a feature that makes the most efficient use possible of a web browser’s local cache. Express Loader tries to reduce or eliminate requests to your site for relatively static content such as images and style sheet (CSS) files.

Use the `Enable Express Loader` checkbox to enable or disable the Express Loader feature. This option is enabled by default. Leave this option enabled unless requested to disable it by Swan Labs Customer Confidence.

When a browser requests content from a web server, the browser places that content in its local cache. If the cached content is not served with an expiration time, the browser makes subsequent requests for that content using a conditional GET. This conditional GET takes the form of an extra request header field such as `If-Modified-Since`. If the requested object appears to be different from the content the browser has cached, a fresh copy of the object is sent to the browser. Otherwise, the browser uses the object cached on its local disk.

Although faster than serving the entire object each time the browser requests it, these conditional GETS can add up to a significant amount of traffic for your site. For sites serving large numbers of images for each page, the client can perceive a slow response time, especially over dialup connections.

However, if the browser receives an expiration time for an object, the browser does not make a conditional GET for subsequent requests of the object. The browser uses its cached object whenever needed as long as that object has not expired and your site sees no subsequent requests for that object once it has been placed in the browser's cache.

Note: If you enable Express Loader, any Browser Cache Minimum age settings that might apply to express loaded objects is ignored. However, the HTML page into which those objects are being loaded honors Browser Cache Minimum Age, if set. The browser age setting is described in "Browser Cache Minimum Age" on page 131.

Link Management

The WebAccelerator takes advantage of a browser's local cache by:

- serving qualifying content with a very long expiration time. This expiration time is long enough that it is unlikely the browser re-requests the content at any time in the near future.
- using a `pv` tag to append a unique value to every such qualifying link or URL embedded in your web pages. This value is a hash of the object and as such is guaranteed to uniquely identify the corresponding content stored by the WebAccelerator.

This activity is performed only these types of links and URLs:

- image tags:
  `<img src="...">`
- script tags:
  `<script src="...">`
- link tags:
  `<link href="...">`
- forms whose input type is an image:
  `<form><input type="image" src="...."></form>`

Express Loader is performed on links of these types only if these conditions are true:

- the links do not contain any query parameters. Even a trailing question mark ('?') causes Express Loader to not manage the link.
- there are no Content Variation rules defined for the links. That is, if the link matches to a policy set that identifies a cookie as being significant for content, then Express Loader is not used for that link.
- there are no proxy rules defined for the link

Express Loader Example

Suppose your site serves a very simple page that consists of a couple of image files. The page that your site serves might look like:

```
<html>
<head><title>example page</title></head>
<body>
      <p>The images that your site serves:</p>
      <p><img src=”myImages/image1.jpeg”></p>
      <p><img src=”myImages/image2.jpeg”></p>
</body>
</html>
```

When Express Loader is enabled, the WebAccelerator modifies and then serves this page in this way:

```
<html>
<head><title>example page</title></head>
<body>
      <p>The images that your site serves:</p>
      <p><img src=”myImages/image1.jpeg;pvRG2076ND”></p>
      <p><img src=”myImages/image2.jpeg;pv7YW905BV”></p>
</body>
</html>
```

The `pv` tag appended to each `src` URL is a hash of the image that the WebAccelerator has stored in its cache. In addition, the browser receives a very long expiration time for each of the image files. As a result, the browser performs subsequent requests for the page by:

- performing a conditional GET for the base page
- obtaining the embedded images directly from its cache

When an image on the page is modified, the page that the WebAccelerator serves changes because the corresponding `pv` tag for that image has changed. Because the page has changed, the client is informed of this event when it performs a subsequent conditional GET of the base page. Upon receiving the refreshed page, the browser notices that a different image is being used (for example, `image1.jpeg;pv4RR87M90` instead of `image1.jpeg;pvRG2076ND`), and it requests that image. The WebAccelerator serves that image with the very long expiration time, allowing the cycle to continue.

Express Connect Management

Use the `Enable Express Connect` checkbox to enable or disable the Express Connect feature. This option is disabled by default. You should leave this option disabled unless your host map for the application that uses this policy specifies Express

Connect options. For information on these options, see “Managing Express Connect” on page 34.

Express Connect is a feature that the WebAccelerator uses to make the most efficient use possible of a web browser’s TCP connections. The WebAccelerator tries to increase the number of concurrent content requests for accelerated web browser page rendering performance.

When a browser requests content from a web server, the browser normally creates at most two TCP connections for each domain. All requests to the domain share these connections and only one request and response is sent on a connection at a time. If the browser is near the web server, the time to perform this request/response handshake is small. However, the client can perceive a slow response time if there is significant network latency between the browser and the server, due to geographic distance or network hops. This can be the case even when there is sufficient bandwidth.

However, if additional domains are created for the objects on the page, this enables more than the default number of connections. The browser uses these additional connections to retrieve these objects and accelerate overall page response time.

The Express Connect facility enables more connections than the default to be used between the browser and the WebAccelerator. Unlike making changes in client browsers, Express Connect can be managed from a single point of control and it can be employed selectively based on the Request Host and protocol.

Link Management

The WebAccelerator takes advantage of a browser’s TCP connection handling abilities by generating an Express Connect domain for every qualifying link or URL embedded in your web pages. URLs qualify for Express Connect management if their domain and protocol have Express Connect options specified in the host map. For each Express Connect domain generated by the WebAccelerator, the browser opens up to two additional TCP connections.

Express Connect activity is performed only for these types of links and URLs:

- image tags:
  `<img src=”...”>`
- script tags:
  `<script src=”...”>`
- forms whose input type is an image:
  `<form><input type=”image” src=”...”></form>`

Express Connect Example

Suppose your site serves a very simple page (`http://www.site.com/index.htm`) that consists of a number of image files. The page that your site serves might look like:

```
<html>
<head><title>example page</title></head>
<body>
      <p>The images that your site serves:</p>
      <p><img src=”myImages/image1.jpeg”></p>
      <p><img src=”myImages/image2.jpeg”></p>
      <p><img src=”myImages/image3.jpeg”></p>
      <p><img src=”myImages/image4.jpeg”></p>
</body>
</html>
```

The Request Host map for `www.site.com` has Express Connect options set to use a subdomain prefix of `pivia`. The maximum number of Express Connect subdomains is set to 2. In this case, the WebAccelerator modifies and then serves this page in the following way:

```
<html>
<head><title>example page</title></head>
<body>
      <p>The images that your site serves:</p>
      <p><img src=”http://pivia1.www.site.com/myImages/image1.jpeg”></p>
      <p><img src=”http://pivia2.www.site.com/myImages/image2.jpeg”></p>
      <p><img src=”http://pivia2.www.site.com/myImages/image3.jpeg”></p>
      <p><img src=”http://pivia1.www.site.com/myImages/image4.jpeg”></p>
</body>
</html>
```

If a request needs to be proxied to an origin server, the subdomain prefixes are automatically removed by the WebAccelerator before requests are proxied.

Compressing Content

You can use assembly policies to identify content that you want the WebAccelerator to compress when it serves the content. The compression mechanism is gzip-encoding. This compression can improve the performance of your site as perceived by the client requesting the content, especially if that client is operating over a dialup connection. Gzipped content can also reduce your site’s bandwidth costs by reducing the size of the data that you are serving.

There are two options that determine whether a response can be gzipped:

- the value of `compressToClient` for the object type the response was classified under

- the Compress Contents checkbox in Content Assembly Options for the assembly policy on the node to which the request and response was matched

The `compressToClient` value is set in the global configuration fragment file, `globalfragment.xml` and it can override the Compress Contents checkbox. If `compressToClient` is set to `none`, it overrides the setting in the assembly policy and the response is never compressed. If `compressToClient` is set to `pivia`, it overrides the setting in the assembly policy and the response is compressed whenever it is being sent to another WebAccelerator, but never compressed when sent to a browser client. If `compressToClient` is set to `policyControlled`, the Compress Contents checkbox setting is used to determine whether the response is compressed or not.

In addition, if the client is not another WebAccelerator, the client that made the request must indicate it can accept gzip-encoded content by specifying the appropriate values on the `Accept-Encoding` HTTP request header. If the client does not indicate it can accept gzip-encoded content, the WebAccelerator serves the content in its uncompressed state.

In general, enable compression. However, gzipping your site's content does carry a small performance cost for the WebAccelerators. Only gzip content that benefits from compression. For content that does not benefit, such as image formats like gif and jpeg), compression should be disabled. The default value of `compressToClient` for image object types is `none`, so compression is disabled by default for them.

For more information on setting `compressToClient`, see the *Administration Guide*.

Content Assembly on Proxies

When this content assembly option is selected, the WebAccelerator performs assembly on responses that it receives as the result of a proxy to your origin servers before forwarding that response on to the requesting client. This feature is primarily intended to allow all of your content to be compressed and to be managed by Express Loader, even if it is not served from cache.

If this option is not selected, then the WebAccelerator does not modify responses received from your origin servers when it forwards them to the requesting client.

Note: If the response from your origin servers indicates that the response contains ESI assembly instructions, then the WebAccelerator always performs assembly on that response regardless of the value selected for this option. For more information on ESI-encoded responses, see "Processing an ESI Response" on page 98.

Enable this option whenever Express Loader is enabled (see "Express Loader Management" on page 84 for information on that option). You should also enable this option if you are compressing your responses.

Rewrite Engine Scripts

Rewrite Engine scripts can be used to manipulate responses during assembly. For example, the `pdf.code` rewrite script linearizes PDF files. Loading the Rewrite Engine scripts refreshes the script archive and makes the scripts available to the WebAccelerator. Before you can select a rewrite script to include in an assembly policy, it must be loaded. The default rewrite scripts, such as `pdf.code`, are loaded automatically during installation. For information on loading or modifying rewrite scripts, see "Customizing Rewrite Scripts" on page 196.

When application matching maps an HTTP request to a specific node in the Request Type Hierarchy, and that request is proxied to your origin server, the response to that request is rewritten by any script you specified in the assembly policy for that node. The rewrite scripts you specify are not run against:

- responses served from cache, because there is no need: any rewrite script would have been run against the original response before it was cached
- responses that were not matched to a node in the Request Type Hierarchy
- responses that match to an object type which has a rewrite script specified, because that script is run instead
- responses mapped to a node where its assembly policy has `Do Not Rewrite` specified

For information on when your assembly policy rewrite setting is overridden by an object type definition, see "Running Rewrite Scripts" on page 197. This is not common, because most object type definitions do not specify rewrite scripts.

To specify a rewrite script for an assembly policy, go to the Assembly screen in the Policy Editor and choose the node whose responses you want rewritten. Select a rewrite script from the drop-down list in the Response Rewriting section and save the policy. Once you publish the policy, any responses that match this node are rewritten by the script.

Advanced Assembly Options

There are some advanced assembly options that enable you to override system settings for a node. You specify the options in a text box on the Assembly screen. This text box is reserved for these special strings, and they must be entered exactly as shown.

The strings are commands that enable features for the node which are normally disabled for the system. The features apply to any requests or responses which match this node. The strings you can specify here are:

Valid Strings	Description
`authAdapter=`*`module`*	where *`module`* is the name of an authentication adapter module you want the WebAccelerator to load and run for requests/responses that match this node. The default is that no module is specified.

`followRedirects=true`
`followRedirects=false`
	when set to `true`, causes the WebAccelerator to internally follow redirects unless certain conditions, such as protocol or domain changes, are met. The system-wide default is `false`.

`forceLastMod=true`
`forceLastMod=false`
	when set to `true`, instructs the WebAccelerator to attempt to generate a Last-Modified header for the response. The system-wide default is `false`.

`disableContentBasedIdentity=true`
`disableContentBasedIdentity=false`
	when set to `true`, disables the content-based identity feature for this node. The system-wide default is `false`, which means that if content-based identity is in effect, it is not disabled for the node.
	For more information on content-based identity, see the *Administration Guide*.

Because the system-wide default for most of these features is `false`, you only need to set these features to false if you are overriding an inherited true setting from parent node.

Defining Assembly Policies

You manage assembly policies using the Policy Editor:

1. Log into the Admin Tool.

2. Go to the Policies screen.
3. Click the Edit link for the policy set whose assembly policies you want to change.
4. Select the Assembly link.

For details on the screens you use to define assembly policies, see the online help.

Parameter Value Substitution Parameters

The parameters that identify parameter substitution rules specify a source and a target for that parameter. They can also optionally provide a URL prefix that limits the URLs on which the WebAccelerator performs the substitution.

Substitution Source

The source of the substitution is where you get the value that is put into the embedded URL in place of the cached value. Usually the source is a specific request element, such as a particular query parameter, and its value is used during substitution. However, you can specify other sources, such as a random number.

When you specify the source, you identify the parameter by data type and name or location in the request. For example, if the source is a query parameter, you specify its name. If the source is an unnamed query parameter, you specify its ordinal, which defines its location in the request URL.

The HTTP Request Data Type Parameters that you can use as the source of a substitution are:

- Query Parameter
- Unnamed Query Parameter
- Path Segment

Or you can use either a:

- number randomizer, described in "Random Value Substitution" on page 84, or
- request URL

You can use the request URL as the source, in which case the entire request URL is used as the value to be substituted. For example, suppose that the request URL is:
`http://www.somesite.com/apps/something.jsp?entity=orange`

The target for the substitution is the `url` query parameter in the embedded URL of the cached page. After substitution, the embedded URL as served by the WebAccelerator looks like:

```
<a href="http://www.somesite.com/anotherthing.jsp?
url=http%3A%2F%2Fwww.somesite.com%2Fapps%2Fsomething.jsp?
entity=orange&....">
```

Substitution Target

The target of the substitution is where you put the value from the source into the embedded URL. The target is always a specific request element, such as a particular query parameter, and its value is replaced during substitution.

When you specify the target, you identify the parameter by data type and name or location in the request. For example, if the target is a query parameter, you specify its name. If the target is an unnamed query parameter, you specify its ordinal, which defines its location in the embedded URL.

Although you can specify the same request element for both the source and the target, the parameter specified for the source is located in the request URL and the parameter specified for the target is located in the embedded URL in the cached page.

The HTTP Request Data Type Parameters that you can use as the target of a substitution are:

- Query Parameter
- Unnamed Query Parameter
- Path Segment

Target URLs

You can limit which URLs embedded in a page are targeted for substitution. By default, the substitution is performed on all embedded URLs in which the identified target appears. Optionally, you can limit this set of URLs by specifying a prefix that an embedded URL must match before the substitution is performed.

For example, if you identify the target of the substitution as the `entity` query parameter, and you limit the substitution to embedded URLs that match:
`http://www.asite.com/valueList.jsp`

then the WebAccelerator performs substitution on this embedded URL:
`http://www.asite.com/valueList.jsp?entity=Larry`

but not on this one:
`http://www.asite.com/apps/lookup.jsp?entity=Moe`

Content Assembly and the Request Type Hierarchy

Assembly policies are organized using the Request Type Hierarchy. The Request Type Hierarchy is described in Chapter 5, The Request Type Hierarchy.

Each leaf node in the Request Type Hierarchy has none or one assembly policy defined for it. This policy has at least one of these defined for it:

- at least one Parameter Value Substitution rule
- whether Express Loader is to be used
- whether Express Connect is to be used
- whether assembly is to be performed when servicing requests that forced a proxy to your origin servers
- whether compression is to be used

The set of parameter value substitution rules defined for a leaf node are a combination of the rules defined locally at the node, and the rules inherited from the node's ancestors. For information on inheritance support in the Request Type Hierarchy, see "Hierarchy Inheritance" on page 57.

Content Assembly Example

Suppose your site receives the types of requests described in "Request Type Hierarchy Example" on page 60 and your Request Type Hierarchy looks like:

```
-Applications
  -Default
  -Search
-Home
-Images
```

Also, suppose your applications are all requested with a query parameter named `sessionID`. You need to make sure that the value set for `sessionID` on the request is used for the `sessionID` query parameter that is used in the URLs embedded in your web pages.

Tasks

In this situation, you should turn on compression for those pages that benefit from compression. This is true of any request that matches to your Home node, or for any node under your Applications node. However, compression should not be turned on for the Images node because graphics do not benefit from compression.

In addition, you should create a parameter value substitution policy based on the `sessionID` query parameter. You should also do this for your `Home` and `Applications` nodes.

Actions

This procedure uses the `Applications` node. You must repeat this procedure for the `Home` node in order for the policies to apply to requests that match to that node.

Note: You do not have to set any assembly policies to your `Images` node. The only thing needed for this node is that gzip-encoding is turned off. Because gzip-encoding is turned off by default, you do not have to explicitly set any assembly policies for the `Images` node.

1. Select the `Applications` node in the Request Type Hierarchy.
2. Click the Add button in the Parameter Value Substitution table.
3. In the Add Parameter Value Substitution screen, go to the Source table and select Query Parameter from the Type drop-down list.
4. Enter `sessionID` to the name field in the Source table to identify the query parameter.
5. Do the exact same thing in the Target table.
6. Click the Save button.
7. You are now on the Assembly page. Note that the parameter that you defined is now showing in the Parameter Value Substitution table.
8. Ensure that the compression checkbox is checked.
9. Click the Save button.

You are done creating this policy. You must publish your policies in order for this change to be used by the WebAccelerator. See "Publishing Policies" on page 45 for more information.

Using the Debugger

The WebAccelerator provides a debugger that can help you understand how your site is behaving relative to the WebAccelerator's assembly policies. The debugger provides two fundamental modes of operation:

- markup

  In markup mode the WebAccelerator provides a color-coded display of the HTML and ESI markup as served by the WebAccelerator. This information also shows you what parameter substitutions the WebAccelerator might have performed if it had performed assembly. Finally, the debugger flags ESI content, if any, that contains errors (both syntax and logical errors are shown).

- ESI

  In ESI mode, the WebAccelerator performs all assembly and then serves up the page exactly as it does without the use of the debugger. However, the final page is served with the ESI statements that were actually used to include content. For example, if your original page source had a complicated esi:try block, then the debugger places in the final (post-assembly) page's source the actual ESI statement that was finally reached during ESI evaluation. Only this statement remains – the remainder of the esi:try block is removed from the page's source as is normally done during ESI evaluation.

In addition to markup and ESI mode, you can also cause the WebAccelerator to always proxy requests that it receives. This is intended as a convenience feature. You can modify the code on your site and efficiently observe the results of those changes as used under the WebAccelerator.

Enabling the Debugger

You enable the debugger when you publish your policies. To enable debugging, access the Policy Editor and select the Publish link.

Check the Enable Debugger box before publishing your policies to turn on the debugger.

Accessing the Debugger

Once debugging is enabled, you can access the debugger by accessing the Policy Editor and selecting the Publish link. Next, select the Debugger button.

In the resulting debugger pop-up screen, choose at least one of the debugging options. Note that if `Markup` is selected, then `ESI` is ignored. For a definition of the debugging options, see "Using the Debugger" on page 95.

Once you have made your selections, click the Generate Code button. The pop-up screen then displays a time-sensitive query parameter and value.

You can use this query parameter along with a query to your site. For example:

```
http://www..somesite.com/apps/doSomething.jsp?_pv38e59927=markup,proxy
```

Instead of the page that the WebAccelerator normally serves, it replies with the debug output.

Chapter 9
Assembling Content with ESI

Processing an ESI Response
Serving ESI Content from the Cache
ESI Templates
ESI-Included HTML Fragments
ESI Language Definition
Parameter Value Substitution
ESI Example

Edge-Side Include (ESI) is an open standard used to control the behavior of web surrogates such as the WebAccelerator. For WebAccelerator content assembly activities, ESI provides a specification for an XML-based language that is used to include content within pages assembled by web surrogates. The WebAccelerator supports the entire ESI language specification, version 1.0. This specification can be found at the ESI website (http://www.esi.org/).

When you use ESI, you place `esi:include` or `esi:inline` statements in your HTML markup. These statements identify the source of HTML fragments for the WebAccelerator to retrieve and incorporate into the pages that it is serving.

ESI provides conditional statements and comparison operators that allow you to conditionally include fragments in the assembled page. You can use these conditional statements to switch the fragments that are included for any given request based on conditions that exist on the requesting URL.

The HTML fragments identified by the `esi:include` statements can themselves be cached by the WebAccelerator. You can use the same content assembly mechanisms to specify the WebAccelerator's caching behavior towards them as you use for any web page. More, HTML fragments included using ESI can each have a different cached lifetime identified for them. Because there are other mechanisms that can be used to build web pages based on pre-defined HTML fragments, the ability to identify different cached lifetimes for HTML fragments is arguably the best reason to use ESI with the WebAccelerator.

Processing an ESI Response

When the WebAccelerator receives an HTTP request that it cannot service from cache, it proxies that request to your origin servers. At that time, it includes an ESI `Surrogate-Capabilities` header in the HTTP request. This request header identifies the WebAccelerator as being an ESI 1.0 compliant device and identifies its device token as `pivia`:

```
Surrogate-Capabilities: pivia=”ESI/1.0”
```

When your origin servers respond to the WebAccelerator’s request, the response must contain an ESI `Surrogate-Control` response header, and this header must identify at least `ESI/1.0` using the content directive:

```
Surrogate-Control: content=”ESI/1.0”
```

If the WebAccelerator does not see `ESI/1.0` identified in the response header, it does not expect there to be ESI markup included in the response. In that event, the WebAccelerator ignores any ESI markup that appears in the response.

In addition to the content directive, your site can also choose to control the cached lifetime of this response by using the max-age or no-store directives. See “ESI Surrogate-Control Headers” on page 215 for more information.

Upon identifying the response as using ESI, the WebAccelerator processes the ESI markup included in the response. As it finds `esi:include` tags that it must process, the WebAccelerator requests the HTML fragments identified by those tags. The request and response proceeds identically to the manner described here for the original request. When the WebAccelerator has received all of the HTML fragments required to service the response, it assembles the final page and uses it to respond to the original request.

Caching the Response

When the WebAccelerator has finished responding to the original HTTP request, it compiles the response received from your origin servers. This compiled response includes the HTML fragments and ESI tags, but not the HTML fragments that the WebAccelerator eventually uses to assemble the final page.

In addition, the WebAccelerator compiles each of the HTTP fragments into the compiled response. These compiled responses are cached according to the caching policies to which they match. Any ESI or HTTP Cache-Control response headers received with each fragment is also used to determine the cache-ability and cached lifetime of the fragment. The ESI and HTTP response headers that can be used to control a response’s cache-ability and cached lifetime are described in “Content Lifetime Mechanisms” on page 132.

Note that in order to cache a fragment, the WebAccelerator must consider the fragment to be complete. See "Caching HTTP Responses" on page 193 for more information.

Caching Fragments

HTML fragments referenced by `esi:include` tags may be obtained from a URL identified on the `esi:include`, or the fragments can be embedded directly in the response using `esi:inline` tags (see "esi:inline" on page 108). Either way, fragments required by the page are obtained by the WebAccelerator and cached separately from the ESI template that referenced them (the template is the body of the document that was sent in the response – see "ESI Templates" on page 99 for more information).

If `esi:inline` is not used, then the fragment is obtained directly from the URL identified on the `esi:include` statement (the WebAccelerator makes an HTTP query to obtain the fragment). If `esi:inline` is used, then the fragment is extracted from the response itself and cached separately from the template.

Serving ESI Content from the Cache

When the WebAccelerator serves content from its cache, it assembles that content using instructions found in the compiled responses that it has cached. When ESI is in use, this assembly involves:

1. Evaluating any ESI variables that appear in the ESI mark-up flow of control. ESI variables are described in "ESI Variables" on page 104.
2. Executing any ESI-include statements found in the templates and fragments needed to service the request. ESI templates are described in the next section. ESI fragments are described in "ESI-Included HTML Fragments" on page 100.
3. Performing parameter value substitution according to the assembly policies in effect for these templates and fragments.

ESI Templates

HTML pages that use ESI markup are considered to be ESI templates. These templates use the same HTML, javascript and other client-side markups as are used by any traditional web page. However, sections of the final HTML do not appear in the markup produced by your origin servers. Instead, ESI tags are used to identify HTML fragments and other resources that should be placed in that section of the page. The

ESI tags can also identify the conditions under which the fragment or resource should be included into the page.

When a web surrogate such as the WebAccelerator sees the ESI tags, it removes them from the page and replaces them with the appropriate fragment or resource.

For example, the following is a very simple ESI template:

```
<html>
<head>
      <title>A simple ESI Template</title>
</head>
<body>
<h1>A simple ESI Template</h1>
      <esi:include src=
"http://www.somesite.com/fragments/fragment1.html"/>
</body>
</html>
```

`fragment1.html` might then be:

```
<p>This is a sample fragment</p>
```

Upon receiving a request for the template, the WebAccelerator ultimately responds with this page:

```
<html>
<head>
      <title>A simple ESI Template</title>
</head>
<body>
<h1>A simple ESI Template</h1>
<p>This is a sample fragment</p>
</body>
</html>
```

ESI-Included HTML Fragments

One of the best reasons for using ESI is when you can identify sections of a page that have different cached lifetime expectancies. If you isolate these page sections into separate, unique chunks, then you can use the WebAccelerator's content lifetime and cache invalidation mechanisms to manage them.

Content lifetime is described in Chapter 11, Cache Control and Content Lifetime. Cache invalidation is described in Chapter 12, Invalidating Cached Content.

Each HTML section that is included into the page is called a fragment. Note that fragments can be requested individually from an origin server, or they can be embedded in the template using `esi:inline` tags. See "esi:inline" on page 108 for more information.

HTML fragments can themselves contain ESI markup. Therefore, a fragment can in effect be a template that uses `esi:include` or `esi:inline` to obtain other fragments.

For any given request, there is a limit to the total number of `esi:include` and `esi:inline` statements that the WebAccelerator processes. By default, this number is 50. This value is configurable by the personnel responsible for administering your WebAccelerator installation. The actual value selected for your installation is gated by the speed of your WebAccelerator's processors, the amount of memory available to each of your WebAccelerators, and the amount of traffic experienced by each WebAccelerator.

Note: Because ESI templates are generally valid HTML pages that do not contain matching `<HTML></HTML>` tags, you should turn off matching HTML tags for the purposes of response caching. Normally, an HTML file is not cached if it does not have opening and closing HTML tags. See "Caching HTTP Responses" on page 193 for more information.

ESI Language Definition

ESI provides a complete language that offers operators, variables, conditional statements, and several pre-defined tags. ESI markup is a subset of XML. The markup is valid XML, but not all valid XML usage is allowed in ESI. ESI is more restrictive than XML, and removes options that give more choice. However, this also serves to simplify both the language and usage of the language. Be aware of the following regarding the ESI language:

- ESI is very sensitive to case. ESI tags must be provided in all lower-case. ESI variables must be provided in all upper case. There are no exceptions.
- ESI is also very clear about closing tags. Some tags must use closing tags, others must not. There are no situations when a closing tag is optional.

The WebAccelerator provides support for the entire ESI 1.0 specification.

The ESI open specification is available here:
http://www.esi.org/language_spec_1-0.html

ESI Expressions

ESI allows for conditional processing using the `esi:choose` block (see "esi:choose" on page 111). These blocks make use of test attributes to drive the evaluation of the conditions. Test attributes make use of expressions:

```
operand operator operand
```

The *operand* can be either an ESI variable or a literal. The `operator` can be any of the comparison operators described in "ESI Operators" on page 102.

ESI Literals

Literal values can be either numeric or strings. String literals can be enclosed in quotes or not. Quotes can be either single or double quotes (' ' or " "). They are required if the string contains blank spaces, or if the string contains any characters not in the set [a-zA-Z0-9_.]. If quoted, a string literal cannot contain that same quote in the string. For example, if you wish to use the literal string:

```
Simon says, "duck"
```

then you quote the literal in this way:

```
'Simon says, "duck"'
```

A non-quoted literal that is composed of only digits and (optionally) a single period (.) is interpreted as a number when this is most meaningful for an operator.

Complex Expressions

Complex expressions are allowed using boolean operators. For example:

```
value1 == value2 | value3 < value4
```

evaluates to true if `value1` is equal to `value2` or if `value3` is less than `value4`.

Operand Associations

Operands associate left to right. Parentheses can be used to explicitly identify associations within an expression:

```
(value1 == value2) | (value3 < value4)
```

Expression Negation

Negation of an expression can be achieved using the unary not (!) operator. For example:

```
!((value1 == value2) | (value3 < value4))
```

ESI Operators

ESI operators include comparison, unary negation, logical and, and logical or operators. White space is optional around all operators. Operators are evaluated in the following decreasing order of precedence:

- binary
- comparison
- unary negation

- logical and
- logical or

Binary

Swan Labs extends the ESI specification with the binary + operator. This operator means either addition or string concatenation. If the + operator is used with mixed data types, then string concatenation is performed. For example:

```
123 + 'foo'
```

evaluates to:

```
123foo
```

Comparison

The comparison operators are:

Operator	Definition
==	equal
!=	not equal
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to

If both operands are numeric, the comparison is performed numerically. If either operand is non-numeric, both operands are evaluated as strings. If an operand is empty or undefined, the expression evaluates to false. For example:

Comparison	Explanation
1 == 1	Evaluates to true. The comparison is performed numerically.
1==’1’	Evaluates to true. The comparison is performed as if both operands are strings
(!(1 == 1))== 1	Evaluates to false. The left-hand operand is of type boolean which is non-numeric. Essentially, this comparison is the same as: ‘false’ == ‘1’
1==	Evaluates to false.

Logical Operators

Logical operators include:

Operator	Definition
!	unary negation
&	logical and
|	logical or

Note that Swan Labs extends the ESI specification to allow logical operators to be used as comparators. When any boolean value is required for an operator, and strings are provided for one or more operands, then empty is considered false and non-empty is considered true. For example:

Comparison	Explanation
!(1 == 1)	Evaluates to false.
1=='1' & 4 < 3	Evaluates to false.
1=='1' | 4 < 3	Evaluates to true.
'foo' | 'bar'	Evaluates to true.
''&'bar'	Evaluate to false.
!''	Evaluates to true.

ESI Variables

ESI supports six read-only variables. They are used to obtain values presented on the HTTP request:

Variable Name	Variable Type	Description
HTTP_ACCEPT_LANGUAGE	list	Contains a list of the values that appear on the HTTP Accept-Language request header.
HTTP_COOKIE	dictionary	Contains the cookies and cookie values presented on the HTTP request.
HTTP_HOST	string	Contains the value provided on the HTTP HOST request header.

Variable Name	Variable Type	Description
HTTP_REFERER	string	Contains the value provided on the HTTP REFERER request header.
HTTP_USER_AGENT	dictionary (special)	Contains the value provided on the HTTP USER_AGENT request header
QUERY_STRING	dictionary	Contains the query parameters and their values that appeared on the request header.

Variable names must be uppercase. To reference a variable's value, surround the variable parenthesis and precede it with a dollar sign ($):

```
$(HTTP_COOKIE)
```

ESI variables are evaluated any time they appear between ESI begin and end tags (such as when they appear inside an `<esi:choose>` block, or between beginning and ending `<esi:vars>` tags). In addition, some ESI tags permit ESI variables in attribute values, but most do not. Notably, the `SRC` and `ALT` attributes for `<esi:include>` does permit ESI variables.

ESI variables that appear in any other situation are emitted literally by the WebAccelerator assembly engine.

The `<esi:choose>` block is described in "esi:choose" on page 111. The `<esi:vars>` tag is described in "esi:vars" on page 107. The `<esi:include>` tag is described in "esi:include" on page 107.

Lists

ESI variables identified as lists return a boolean depending on whether the requested value is present. You identify the requested value by appending braces ({}) containing the value that you are requesting to the variable name:

```
$(HTTP_ACCEPT_LANGUAGE{en-gb})
```

returns true if `en-gb` appears on the header.

The list value you specify is case-sensitive.

Dictionaries

ESI variables identified as dictionaries allow you to access a string that is identified by a key. You specify the key by appending braces ({}) containing the key to the variable name:

```
$(QUERY_STRING{action})
```

The key is the name used by an HTTP request element. It is case-sensitive. For example, if you are using the QUERY_STRING variable, the key is a query parameter name. The value returned is the value presented for the requested HTTP request element. If the requested element did not appear on the URL, an empty string is returned.

So for this URL:
`http://www.somesite.com/srch.jsp?action=forward&sessionID=AAYu30Ws`

the QUERY_STRING variable returns:

Variable	Result
$(QUERY_STRING{action})	'forward'
$(QUERY_STRING{sessionID})	'`AAYu30Ws`'
any other key	''

Default Values

An empty string is returned whenever a value is not found for a variable access. That is, if the variable's value is empty, if a dictionary key is undefined, or if the variable is non-existent, the result of the access is an empty string ('').

You can use the logical or (`|`) operator to identify a default value for a variable in the event that it returns an empty string:
`$(VARIABLE|default_value)`

For example:
`$(HTTP_COOKIE{userID}|'0')`

returns either the value set for the `userID` cookie, or `'0'` if `$(HTTP_COOKIE{userID})` evaluates to an empty string.

Cookie Handling

The WebAccelerator examines cookies presented in the request as per the ESI specification. However, if a request forces the WebAccelerator to proxy for fresh content, and if the response that the WebAccelerator receives from that proxy includes Set-Cookie directive(s), then the original request is processed as if the request presented those cookies. This allows your cookie-based ESI scripts to operate on the request even if the necessary cookie(s) were not originally presented on it.

esi:vars

Used to include an ESI variable in markup outside an ESI block. For example:

```
<esi:vars>
      <p>Currently logged in as $(HTTP_COOKIE{userID}|'anonymous')</p>
</esi:vars>
```

esi:include

```
<esi:include src="uri" [alt="uri"] [onerror="continue"]/>
```

Used to identify a fragment that the WebAccelerator should include in the final assembled page. This tag is an empty element – you must not provide a closing tag for it.

You are limited in the number of `esi:include` tags that the WebAccelerator processes for a given client request. See "ESI-Included HTML Fragments" on page 100 for more information.

This tag accepts these arguments:

Argument	Definition
`src`	Required argument that identifies the resource the WebAccelerator is to include into the assembled page. You must provide a valid URI for this argument. If you provide a relative URI, then the WebAccelerator resolves the URI relative to the requested template.
`alt`	Optional argument that identifies an alternative resource. This resource is used in the event that the resource identified by the `src` argument cannot be obtained. You must provide a valid URI for this argument. If you provide a relative URI, then the WebAccelerator resolves the URI relative to the requested template.
`onerror`	Optional argument that forces the WebAccelerator to silently continue processing in the event that it cannot obtain the resource identified by the `src` and `alt` arguments. If this argument is not provided, then failure to obtain the identified resources results in the WebAccelerator returning a 400-series status code and message. Note that if the `esi:include` statement is placed in an `esi:try` block (see the next section), then the `esi:try` block prevents the 400-series status code from being returned to the client.

For example:

```
<esi:include src="http://www.somesite.com/includes/frag1.html" />
<esi:include src="frag1.html" alt=
"http://web2.somesite.com/includes/frag1.html" />
<esi:include src="../images/sunset.jpg" onerror="continue" />
```

esi:inline

```
<esi:inline name="uri" [fetchable="yes|no"] [src="uri"] [max-age=int]>
   ... html fragment ...
</esi:inline>
```

Used to demarcate a fragment that is referenced by an `esi:include` statement on the page. You may place as many `esi:inline` statements on the page as are required by the remainder of your ESI markup.

If the fragment identified by the `esi:inline` statement is marked as non-fetchable (`fetchable=no`), then upon expiration the fragment is fetched by refreshing the page which contains the fragment. If the fragment is marked as fetchable, then a source URI for that fragment must be provided. In the event that the fragment expires, it can be refreshed from the source URI without refreshing the containing page.

The first time a page containing an `esi:inline` statement is accessed, the HTML fragment delimited by the `esi:inline` statement is cached. Subsequent requests for that HTML fragment are serviced from the cache, even if that fragment is requested by some other page using either an `esi:inline` or `esi:include` statement..

`esi:inline` statements are important because they allow you to add ESI support to your site without re-architecting it to store and serve HTML fragments. Essentially, by marking up the appropriate sections of your site's pages with non-fetchable `esi:inline` statements, you can cache and re-use the HTML fragments needed by your site.

This tag accepts these arguments:

Argument	Definition
`name`	Required argument that provides a URI which uniquely identifies the fragment. In the event that the fragment is fetchable, and if the fragment's expiration time is different than its template's expiration time, then this URI is used to fetch the fragment when it is refreshed.
`fetchable`	Required argument that identifies whether the fragment can be obtained from the URI identified by the `name` parameter. Valid values are either `"yes"` or `"no"`. Note that `fetchable="yes"` only has meaning if the fragment's cache expiration time is different than the template's expiration time. If the two expire at the same time, then the fragment is always refreshed when the template is refreshed. If the fragment is fetchable and if the fragment's expiration time is different than the template's expiration time, then the WebAccelerator refreshes the fragment using the URI identified on the `src` parameter.

Argument	Definition
`src`	Required argument if `fetchable` is `"yes"`. If the fragment's expiration time is different than its template's expiration time, then this parameter is used to identify the URI where the fragment can be obtained. At a minimum, this argument should be set to the URI used to obtain the template. If `fetchable="no"`, then this argument is ignored and the URI provided on the name attribute is used. Note that this parameter is a Swan Labs extension to the ESI specification.
`max-age`	Optional argument that identifies a cache expiration time for the fragment. Units are in seconds. In order for this parameter to be obeyed, the WebAccelerator must have a lifetime policy set for the template such that ESI max age is obeyed. For more information on lifetime policies, see Chapter 11, Cache Control and Content Lifetime. Note that this parameter is a Swan Labs extension to the ESI specification.

For example:

```
<esi:inline name="http://www.somesite.com/includes/frag1.html"
      fetchable="no" max-age=300>
      <p>This is a sample HTML fragment that is included into
      this page when the appropriate esi:include statement is evaluated.
      </p>
</esi:inline>
<esi:include src="http://www.somesite.com/includes/frag1.html" />
<esi:inline name="http://www.somesite.com/includes/frag2.html"
      fetchable="no" max-age=250>
      <p>This is yet another sample HTML fragment that is
      included into this page when an esi:include statement is evaluated.
      </p>
</esi:inline>
<esi:include src="http://www.somesite.com/includes/frag2.html" />
```

esi:try

```
<esi:try>
      <esi:attempt>
      ...
      </esi:attempt>
      <esi:except>
      ...
      </esi:except>
</esi:try>
```

Used to perform exception handling for `<esi:include>`. Each `<esi:try>` block must have exactly one `<esi:attempt>` block and exactly one `<esi:except>` block. No other children of `<esi:try>` are allowed. `<esi:try>`, `<esi:attempt>`, and `<esi:except>` must all be terminated by their respective closing tags.

When processing an `<esi:try>` block, the WebAccelerator first attempts to process the markup contained in the `<esi:attempt>` block. If the `<esi:include>` found there triggers an error, then the WebAccelerator processes the markup contained in the `<esi:except>` block.

Note: If `<esi:include>` is used with the `onerror=”continue”` argument, the WebAccelerator never processes the contents of the `<esi:except>` block. This is because the WebAccelerator suppresses any error conditions that might arise when processing an `<esi:include>` if `onerror=”continue”` is also used.

The WebAccelerator replaces the entire `<esi:try>` block with the results of the markup that it processes, regardless of whether that markup is found in the `<esi:attempt>` block or the `<esi:except>` block.

The markup contained by an `<esi:attempt>` or `<esi:except>` block can be any valid markup used by your web pages. For example, if your site’s pages are comprised of HTML and javascript, then any valid combination of HTML, javascript, and ESI markup could appear there.

For example:

```
<esi:try>
      <esi:attempt>
      <p>On sale:</p>
      <esi:include src=”http://images.mysite.com/vase.html”
      alt=”http://images.mysite.com/overstock.html” />
      </esi:attempt>
      <esi:except>
      <p>We’re sorry! Our image server appears to be unreachable. Please
      try again at some later date.</p>
      </esi:except>
</esi:try>
```

esi:choose

```
<esi:choose>
      <esi:when test=”...”>
      ...
      </esi:when>
      <esi:when test=”...”>
      ...
      </esi:when>
      <esi:otherwise>
      ...
      </esi:otherwise>
</esi:choose>
```

Provides conditional processing of the ESI block. An `<esi:choose>` block has zero or more `<esi:when>` blocks, and zero or one `<esi:otherwise>` block.

The markup contained between the `<esi:when></esi:when>` tags is processed by the WebAccelerator when the `test` argument evaluates to true. The WebAccelerator processes the first `<esi:when>` block whose test evaluates to true and then exits the `<esi:choose>` block. The WebAccelerator replaces the entire `<esi:choose>` block with the results of processing that markup.

The test provided for an `<esi:when>` block must be a valid ESI expression. See “ESI Expressions” on page 101 for more information.

If none of the tests provided for the `<esi:when>` blocks evaluates to true, the WebAccelerator processes the markup contained in the `<esi:otherwise>` block, if it is defined. The WebAccelerator replaces the entire `<esi:choose>` block with the results of processing that markup.

If none of the tests provided for the `<esi:when>` blocks evaluates to true, and there is no `<esi:otherwise>` block defined for the `<esi:choose>` block, the WebAccelerator removes the entire `<esi:choose>` block (replacing it with nothing) from the final assembled page.

The markup contained by an `<esi:when>` or `<esi:otherwise>` block can be any valid markup used by your web pages. For example, if your site’s pages are comprised of HTML and javascript, then any valid combination of HTML, javascript, and ESI markup could appear there.

For example:

```
<H1>Plan Options</H1>
<esi:choose>
      <esi:when test=”$(HTTP_COOKIE{membership}==’gold’>”
      <p>Congratulations on your gold membership! Choose from one of the
      following:</p>
      <esi:include src=”http://somesite.com/apps/planOptions.jsp?plan=1”>
      </esi:when>
      <esi:when test=”$(HTTP_COOKIE{membership}==’silver’>”
      <p>Please choose from one of the following silver membership
options:</p>
      <esi:include src=”http://somesite.com/apps/planOptions.jsp?plan=2”>
      </esi:when>
      <esi:otherwise>
      <p>You are not currently a member!</p>
      <esi:include src=”http://somesite.com/apps/planSignUp.jsp”>
      </esi:when>
</esi:choose>
```

esi:comment

`<esi:comment text=”a comment string” />`

Used to comment ESI instructions. The WebAccelerator removes these from the final page that it assembles.

This tag is an empty element – you must not provide a closing tag for it.

esi:remove

```
<esi:remove>
      ....
</esi:remove>
```

Used to provide non-ESI markup. This markup is removed by the WebAccelerator when it performs ESI processing on the page.

The markup contained in the `<esi:remove>` block is intended to be rendered by the client in the event that the WebAccelerator does not perform ESI processing on the page. In this case, the client ignores all of the ESI markup because it represents tags that it does not understand. The contents of the `<esi:remove>` block, however, should be markup that the client understands and so it is rendered by the client.

The WebAccelerator fails to process ESI markup on a page if the appropriate ESI Surrogate-Control response header is not sent in the response to the WebAccelerator. See “Processing an ESI Response” on page 98 for more information.

For example:

```
<p>This hour’s hottest investment tip:</p>
<esi:include src=”investTip.jsp” />
<esi:remove>
      <p><em>Buy low and sell high.</em></p>
</esi:remove>
```

<!--esi ... -->

```
<!--esi
      ...
-->
```

or

```
<--'
      ...
'-->
```

The `<!--esi` ... `-->` construct can be used to encapsulate ESI markup in the template. The WebAccelerator removes the `<!--esi` and `-->` tags from the markup before sending the assembled page to the requesting client. Any information found between these tags is sent to the client.

This construct is optional. ESI tags should be ignored by any client that does not understand ESI. Primarily you should use this construct to guard against a client rendering valid HTML that may be incomplete if the corresponding ESI tags are not executed by a web surrogate. For example, a statement in like this:

```
<esi:vars>
      <p>Currently logged in as $(HTTP_COOKIE{userID}|'anonymous')</p>
</esi:vars>
```

may be safer if it is expressed like this:

```
<!--esi
      <esi:vars>
      <p>Currently logged in as $(HTTP_COOKIE{userID}|'anonymous')</p>
      </esi:vars>
-->
```

These rules represent Swan Labs's interpretation of the ESI specification relative to comment handling:

- any ESI markup, most notably, ESI includes, found inside the `<!--esi` ... `-->` are processed by the WebAccelerator before the page is sent to the requesting client
- any ESI markup found inside normal HTML comments (<!-- ... -->) is also processed by the WebAccelerator before it is sent to the requesting client

In addition, Swan Labs extends the ESI specification by adding the <--' .. '--> construct. Any ESI tags found within this construct are completely ignored by the WebAccelerator.

Parameter Value Substitution

Before you read this section, you should read “ESI Templates” on page 99, “ESI-Included HTML Fragments” on page 100, and gain some familiarity with the ESI language as described in “ESI Language Definition” on page 101.

ESI templates and the HTML fragments are both obtained from your origin servers using HTTP requests in much the same way that any content is obtained from your site. Depending on how you built your site and the matching policies that you defined for the WebAccelerator, different caching policies can be in effect for the templates and fragments needed to service any given request. This has important implications when it comes to parameter value substitutions.

The WebAccelerator determines what portion of an HTML page/template/fragment is the target of a substitution when that object is compiled into a response. The rule for how this substitution is performed is also stored with the response at compile time.

Request time (assembly time) is when the actual substitution is performed. The substitution is performed based on information found in the HTTP request for the ESI template. However, the parameter value substitution rules defined for the ESI template are not necessarily relevant to the substitutions performed for the included fragments.

For example, suppose you have an application matching node that matches to ESI templates, and another that matches to HTML fragments included by those templates:

```
-Applications
-Templates
-Home
-Images
```

This is a typical scenario because it allows you to have differing cache lifetimes for the two types of data.

Suppose the Applications node defines a parameter substitution rule that causes substitution to be performed on query parameter `IDENT`. The source of the value for `IDENT` is the `IDENT` query parameter found on the HTTP request. The templates node, however, does not have this rule defined for it. Instead, it has a random number substitution rule defined for the IDENT query parameter.

Now suppose the request for the template is:

```
http://www.somesite.com/apps/dosomething.jsp?IDENT=3798
```

Then in this case, any embedded URLs that originated in the template have `IDENT` set to 3798. Any embedded URLs that originated in the fragments that are included by the template have some random number set for the `IDENT` query parameter (say, 2290).

These mismatched values appear in the final page served by the WebAccelerator. Depending on how your site is architected and on what the respective embedded URLs are actually used for, this may or may not be the desired behavior. For this reason, you should pay careful attention to how your assembly policies are defined when you are mixing them with ESI markup.

ESI Variables

Parameter value substitution is never performed on HTML tags where ESI variables are used. Parameter value substitution is also never performed inside an ESI tag.

Suppose you have a parameter value substitution rule that causes the value of the URL query parameter to be replaced with some value found on the request URL. Then consider the following embedded URL:

```
<a href="http://www.somesite.com/dosomething.jsp?URL=$(HTTP_REFERER)">
```

Because this URL uses an ESI variable – `$(HTTP_REFERER)` – the parameter value substitution is ignored. Instead, the ESI variable is evaluated and its value is used for the `URL` query parameter.

Now, suppose the same parameter value substitution is defined for a page in which the following include statement appears:

```
<esi:include src="http://www.somesite.com/fragments/firstBlock.jsp?
URL=http://www.somesite.com" />
```

Again, in this case the parameter value substitution rule is ignored. The reason why is that the URL query parameter appears in an ESI tag. In this situation, in order to have the value of the URL query parameter substituted to a meaningful value at assembly time, use an ESI variable:

```
<esi:include src="http://www.somesite.com/fragments/firstBlock.jsp?
URL=$(QUERY_STRING{URL})" />
```

ESI Example

Suppose your site offers stock quote information from stock exchanges all around the world. Further, your site has a policy of updating stock quotes only once every 15 minutes, so while a given stock exchange is active you want quotes from that exchange to have a cached lifetime of 15 minutes. However, when a given stock exchange is not active, you want quotes from that exchange to stay cached until trading reopens. In this way, your origin servers only see request loads for stock exchanges that are in active trading.

You present stock quotes to your customers using a series of tables, each of which contains requested information for a given stock exchange. To manage the varying nature of the content requested by your users, your site generates a unique ESI template for each user. This ESI template is usually fairly stable, so you want it to have a cached lifetime of 12 hours (if the user changes the information that he wants to see, you can use ESI cache invalidation to invalidate her template from the cache – see "Raising Invalidation Rules with ESI" on page 152 for more information).

Also, the ESI template is personalized for each user based on the information found in the `familiarName` and `surname` cookies. In addition, the `display` cookie is used to identify whether the user wants the `price` or `volume` displayed. Finally, the actual stocks to be displayed, and the exchanges where they are listed, are tracked as a database query keyed to the user's ID. This user ID is tracked using the `userid` cookie.

For each fragment included into the template, you explicitly set a cache TTL using the HTTP `Expires` response header (see "HTTP/1.1 Cache-Control Headers" on page 133 for more information).

Requests for stock quote information at your site are of the form:
`http://stocks.somesite.com/quotes/quotes.jsp`

Requests for a quote from a given exchange are expressed inside `esi:include` tags. The URLs used for these requests are of the form:
```
http://stocks.somesite.com/quotes/showQuotes?display=volume&
exchange=NASDAQ&user=820o8xdmp22
```

Request Type Hierarchy

Start by setting up the policies for your site. Create two top-level nodes in the Request Type Hierarchy:

Node	Application Matching Parameter
Templates	Create a single parameters based on the Path data type: `/quotes/quotes.jsp`
Quotes	Create a single parameter based on the Path data type: `/quotes/showQuote.jsp`

Caching Policies

For the `Templates` node:

1. Create a lifetime policy that identifies a maximum lifetime of 12 hours. Also, make sure the HTTP Headers lifetime mechanism is turned off for this node. This prevents users from forcing proxies to your origin servers through the use of control-refresh (Internet Explorer) or shift-refresh (Netscape) in their browsers. For information on setting lifetime policies, see "Defining Lifetime Policies" on page 137.

2. Create a variation policy that identifies the `familiarName`, `surname`, `display`, and `user` cookies as being significant for content. For information on variation policies, see Chapter 7, Setting Variation Policies.
3. Create a proxying policy that indicates a must-proxy condition if the `userid` cookie contains an empty value or is absent from the request. For information on proxying policies, see Chapter 10, Setting Proxying Policies.

For the `Quotes` node:

1. Create a lifetime policy that has HTTP Headers turned on. Just to be safe, identify a minimum lifetime of 15 minutes for this node. Also, for the reason outlined above, you should use this policy to disallow no-cache HTTP request headers.
2. Create a proxying policy that indicates a must-proxy condition if the `user` query parameter contains an empty value.

The ESI Template

Suppose the user is obtaining stock quotes from the NASDAQ and from the London Stock Exchange. You know this based on a database query that is run when your origin servers receive a request for quotes from the user. Then the template returned by your servers might look like this:

```
<html>
<head>
    <title>
         <!--esi
         <esi:vars>
         Stock quotes for $(HTTP_COOKIE{familiarName})
             $(HTTP_COOKIE{surname})
         </esi:vars>
         -->
         <esi:remove>
         Stock quotes not processed
         </esi:remove>
    </title>
</head>
<body>
<!--esi
<esi:vars>
<h1>Stock quotes for $(HTTP_COOKIE{familiarName})
    $(HTTP_COOKIE{surname})</h1>
</esi:vars>
<esi:comment text=”get stock info for the NASDAQ” />
<esi:try>
    <esi:attempt>
        <esi:include src=”http://stocks.somesite.com/quotes/
showQuotes.jsp?
         display=$(HTTP_COOKIE{display})&exchange=NASDAQ&
```

```
27:         user=$(HTTP_COOKIE{userid}) />
28:     </esi:attempt>
29:     <esi:except>
30:         <h2>NASDAQ Quotes</h2>
31:         <p>We're sorry, but the quotes you requested are not
32:        available. Please wait a few minutes before trying again.</p>
33:     </esi:except>
34: </esi:try>
35: <esi:comment text="get stock info for the London Stock Exchange" />
36: <esi:try>
37:     <esi:attempt>
38:         <esi:include src="http://stocks.somesite.com/quotes/
39: showQuotes.jsp?
40:         display=$(HTTP_COOKIE{display})&exchange=LSE&
41:         user=$(HTTP_COOKIE{userid}) />
42:     </esi:attempt>
43:     <esi:except>
44:         <h2>London Stock Exchange Quotes</h2>
45:         <p>We're sorry, but the quotes you requested are not
46:        available. Please wait a few minutes before trying again.</p>
47:     </esi:except>
48: </esi:try>
49: -->
50: <esi:remove>
51: <p>Due to an unknown error, your stock quotes could not be retrieved.
52: If the problem persists, <a href=
"mailto:planadmin@somesite.com">please
53: contact us.</a></p>
54: </esi:remove>
55: </body>
56: </html>
```

How it Works

Lines 2 - 14 define the HTTP head section of the page that the WebAccelerator returns to the client. If the WebAccelerator processes the ESI markup in this section of the page, then it returns:

```
<head>
      <title>
      Stock quotes for Jen Babsen
      </title>
</head>
```

Because of the must-proxy policy set for the `userid` cookie, you can safely assume that the `familiarName` and `surname` cookies are set when the WebAccelerator processes this ESI markup. As a result, you do not need to set a default name here.

If the WebAccelerator does not process this ESI markup (because the client manages to request the page directly from your origin servers, or because your origin servers do not respond to the WebAccelerator's request with an appropriate `Surrogate-Control` response header), then the ESI markup appears to be an HTML comment (or unknown tags in the case of the markup on lines 10 and 12) that the client ignores.

Similarly, lines 15 - 55 define the body section of the page. Again, if the WebAccelerator does not process this ESI markup, the contents of the `<esi:remove>` block at lines 50 -54 are the only thing rendered by the client. So if the client somehow receives the unprocessed template, it renders:

```
<html>
<head>
      <title>
      Stock quotes not processed
      </title>
</head>
<body>
<p>Due to an unknown error, your stock quotes could not be retrieved.
If the problem persists, <a href=”mailto:planadmin@somesite.com”>please
contact us.</a></p>
</body>
</html>
```

Lines 24 - 27 and lines 38 - 41 represent include attempts. If the `esi:include` from lines 24 - 27 fails, then the WebAccelerator sends the HTML markup in lines 30 - 32 to the client for rendering. Similarly, if the `esi:include` from lines 38 - 41 fails, then the client receives the HTML markup shown in lines 44 - 46.

If the request identified by the `esi:include` in lines 24 - 27 succeeds, then the WebAccelerator replaces lines 22 - 34 with the fragment that it receives from your origin servers (or from its own cache, if the fragment is available there). The identical thing happens for the `esi:include` shown in lines 38 - 41.

When your origin servers respond to a request for this template, they do not explicitly set any form of a expiration time for the template in cache. This is because you have already set a maximum TTL of 12 hours for the template. In addition, if the user updates her stock tracking preferences in such a way as to change the template (this happens if she added or removed a stock exchange from the template), your origin servers send the WebAccelerator an ESI content invalidation request to force it to expire this template regardless of the template's current TTL in cache. For more information on content invalidation, see Chapter 12, Invalidating Cached Content.

The HTML Fragment

The `esi:include` directives shown in lines 24 - 27 and 38 - 41 in the preceding example results in the WebAccelerator retrieving an HTML fragment. The WebAccelerator replaces the `esi:include` tags (and the surround `esi:try` statements) with the fragment that it receives.

The actual fragment that the WebAccelerator receives is determined by what the `showQuotes` application returns. This is gated by what the `display` and `userid` cookies are set to. Assuming that `display` is set to `price` and `userid` is set to `EV93jmCV00e4`, then the request that the WebAccelerator send is:

```
http://stocks.somesite.com/quotes/showQuotes.jsp?display=price&
exchange=NASDAQ&user=EV93jmCV00e4
```

Upon receiving this request, the showQuotes application performs a database lookup to see what NASDAQ-listed stocks user `EV93jmCV00e4` is tracking. Based on this lookup, the application might return a fragment that looks like the following:

```
<h2>NASDAQ Quotes</h2>
      <table>
      <tr><td>PHONY1</td><td>13.29</td><td>+0.03</td></tr>
      <tr><td>PHONY2</td><td>2.09</td><td>-2.19</td></tr>
      </table>
```

When your origin server responds to the WebAccelerator with the fragment, it uses the HTTP Expires response header to indicate when the WebAccelerator should refresh the fragment by proxying for new content. If the NASDAQ is in active trading, the expires time is set to 15 minutes in the future. If the NASDAQ is currently closed, the expires time is the day and time when the NASDAQ reopens for business.

If the user updates her list of NASDAQ-listed stocks that she wants to track, your origin servers send the WebAccelerator an ESI content invalidation request to force it to expire this fragment regardless of the fragment's current TTL. For more information on content invalidation, see Chapter 12, Invalidating Cached Content. Content lifetime is described in Chapter 11, Cache Control and Content Lifetime.

Chapter 10

Setting Proxying Policies

Supported HTTP Request Data Types ◂
Always Proxy ◂
Proxy Rules ◂
Proxy Override Rules ◂
Proxying and the Request Type Hierarchy ◂

Proxying policies identify the elements on an HTTP request URL that indicate the WebAccelerator should proxy the request to your origin servers, instead of attempting to service it from cache.

Proxying policies can include override rules, which identify conditions under which the WebAccelerator should ignore all proxy rules. When a proxy override rule applies to an HTTP request, the request is eligible for servicing from the cache, even if there is a proxy rule that indicates the WebAccelerator should proxy the request.

Supported HTTP Request Data Types

The HTTP Request Data Type Parameters on which you can base both proxy rules and proxy override rules are:

- Protocol
- Host
- Method
- Query Parameter
- Unnamed Query Parameter
- Path Segment
- Cookie
- User Agent
- Referrer
- Header
- Client IP

Always Proxy

You can create a policy with a rule to proxy any request that matches to a particular leaf node. Select the node and go to the Proxying link. Check the `Always proxy these requests` checkbox and click Save. If you set this for a node other than a leaf node, this setting is inherited by all the children of the node. After you publish this policy, all requests that are matched to leaf nodes with this setting are proxied.

This Always Proxy setting overrides any other proxy rules or proxy override rules that exist for the node. If a leaf node has inherited this setting, you can override it to turn it off.

The Always Proxy setting applies to any request or response that matches this node. If a response matches a node that has Always Proxy set, it is cached with this setting. When a request is received and the Peformance Server attempts to retrieve this response from cache, it sees the Always Proxy setting on the cached response and proxies the request.

Proxy Rules

Under most circumstances, an HTTP request received by the WebAccelerator is eligible to be serviced from cache. Whether the request actually is serviced from cache depends on many things, such as:

- whether the requested content has been served before
- whether the requested content is cache-able
- whether the cached content is fresh or expired

The process that the WebAccelerator uses is described in "Servicing HTTP Client Requests from Cache" on page 192.

There might be types of requests you do not want to be serviced from the cache. A common reason is use of session tracking information. This information might be set in a cookie or a segment parameter. If you are using a session tracking mechanism, any request received without that tracking information should be proxied to your origin servers so the session information can be set. You can use proxy rules to identify these kinds of conditions.

Generally, proxy rules are only relevant to requests that match their node, not to responses that match.

Defining Proxy Rules

You manage proxy rules using the Policy Editor:

1. Log into the Admin Tool.
2. Go to the Policies screen.
3. Click the Edit link for the policy set whose proxy policies you want to change.
4. Select the Proxying link.

For details on the screens you use to define proxy rules, see the online help.

Proxy Rule Parameters

Proxy Rule policies are based on HTTP Request Data Type Parameters. You identify the parameter to look for in a request and specify the value it must have in order for a request to be proxied. If a request matches the rule, the WebAccelerator proxies the request to your origin servers. For example, you can create a proxying rule based on a cookie named `version`. You can indicate that the WebAccelerator must proxy the request if:

- `version` does not appear on the request, or
- `version` appears on the request but has no value set for it (`version` is empty)

Proxy Example

Suppose your site receives the types of requests described in "Request Type Hierarchy Example" on page 60 and your Request Type Hierarchy looks like:

```
-Applications
  -Default
  -Search
-Home
-Images
```

Suppose you use a segment parameter to contain special identifying information for your shopping cart application. Requests for your applications are all of the form:

```
http://www.somesite.com/apps/doSomething.jsp;AAy23BV39
```

If the session tracking string does not appear in the segment parameter at the end of the URI, you want the request to be proxied to your origin servers for special handling.

Tasks

In this situation, you should create a proxy rule for your Applications node. Create this rule with just one parameter based on the Path Segment data type. This rule should identify the subject as being:

- path ordinal 1 in the full path as counted from right to left, and
- segment parameter ordinal 1

For this rule, you want it to be in effect if the `Parameter is EMPTY string` or `Parameter is Absent`.

Actions

To create the proxy rule described above:

1. Select the Applications node in the Request Type Hierarchy.
2. Select the Proxying link.
3. Select Path Segment from the Add Parameter drop-down list in the Proxy Rules table and then click the Add button.
4. In the Add Parameter screen:
   - give the parameter a meaningful alias. This alias appears in the Proxy Rules table in order to help you identify the parameter in the future.
   - identify the segment as ordinal 1 as counted from right to left
   - identify the parameter ordinal as 1

   See "Path Segment" on page 50 for information on how path segments and their ordinals work.

5. In the Value(s) area:
   - check `Value is an EMPTY String`
   - check `Parameter is ABSENT from the request`
6. Click the Save button.

The rule now appears in the Proxy Rules summary table. You must publish your policies in order for this change to be used by the WebAccelerator. See "Publishing Policies" on page 45 for more information.

Proxy Override Rules

Proxy override rules allow you to identify when the WebAccelerator should ignore all proxy rules. If the conditions identified by a proxy override rule are met, the request is eligible to be served from the cache in the same way as any normal request.

Proxy override rules are intended to let you ensure that traffic related to web crawlers and robots is off-loaded from your origin servers. Some sites see a large amount of traffic from these types of clients. If your site is one of these, it is likely that you do not want your origin servers to experience the entire load caused by them. If your proxy rules are based on a cookie that you expect to be set, it is especially important to consider a proxy override rule because web crawlers and robots rarely present cookies in their requests.

Proxy override rules are defined in exactly the same way as proxy rules. The only difference is that you use the Proxy Override Rules section of the Proxying screen to define the rules. See "Proxy Rules" on page 123 for information on how these rules are defined. See the online help for details on the screens that you use to define proxy override rules.

Proxy Override Example

Suppose you use a cookie to contain session tracking information. If the session tracking cookie is not presented on an HTTP request, you want the request to be proxied to your origin servers so that the cookie can be set. As a result, you have created a proxying rule based on the cookie. This rule indicates that the WebAccelerator must proxy any request that does not include the cookie, or for which the cookie is empty.

However, suppose your site experiences a lot of traffic from a web crawler. This web crawler never presents cookies. If you leave the proxy rule in place, your origin servers must handle every request from the web crawler. This places an unnecessary load on your origin servers.

After examining your web server log files, you have discovered that the web crawler presents a string for the HTTP USER_AGENT request header that looks very much like the value presented for MSIE 5.0 browsers. However, the web crawler adds to the USER_AGENT string:

```
badcrawler
```

Tasks

In this situation, add a proxy override rule to the node's proxying policy. Base this rule on the User Agent parameter, to match this regular expression:

```
.*badcrawler.*
```

Actions

To create the proxy override rule described above:

1. Select the Applications node in the Request Type Hierarchy.
2. Select the Proxying link.
3. Select User Agent from the Add Parameter drop-down list in the Proxy Override Rules table and then click the Add button.
4. Check the box for Value, and select MATCHES from the drop-down list.
5. Enter the regular expression that matches the user agent value for the web crawler into the value box:
   ```
   .*badcrawler.*
   ```
6. Click the Save button.

The rule now appears in the Proxy Override Rules summary table. Once published, any request received by your site whose HTTP USER_AGENT value matches the regular expression shown in the proxy override rule is a candidate for servicing from the cache regardless of any proxy rules that may also be defined. See "Publishing Policies" on page 45 for more information on publishing policies.

Proxying and the Request Type Hierarchy

Proxying policies are organized using the Request Type Hierarchy. The Request Type Hierarchy is described in Chapter 5, The Request Type Hierarchy.

Each leaf node in the Request Type Hierarchy has none or one proxying policy defined for it. If a proxying policy is set for a node, it has at least one of these defined for it:

- an `Always proxy these requests` condition. See "Always Proxy" on page 122.
- at least one proxy rule. See "Proxy Rules" on page 123.

- at least one proxy rule or at least one proxy override rule or both. See "Proxy Override Rules" on page 125.

The set of Always Proxy settings, proxy rules, and proxy override rules defined for a leaf node are a combination of rules and settings defined locally at the node and rules and settings inherited from the node's ancestors. If Always Proxy is set for a leaf node, whether the setting was inherited or set at the leaf node, it overrides all the proxy and proxy override rules for the node. For information on inheritance support in the Request Type Hierarchy, see "Hierarchy Inheritance" on page 57.

Chapter 11

Cache Control and Content Lifetime

Content Lifetime Settings ◀
Content Lifetime Mechanisms ◀
Defining Lifetime Policies ◀
Lifetime Mechanism Precedence Rules ◀
Lifetime Example ◀

Each time the WebAccelerator caches web pages, it must decide how long to keep that content. This length of time is called the content's lifetime. Content lifetime is expressed in the form of a Time to Live (TTL) value. This TTL value can vary for each compiled response.

When cached content has been in the cache longer than its TTL value, it is expired. When the WebAccelerator receives a request for expired content, it proxies that request to your origin servers. The WebAccelerator refreshes the expired content with the new content from your origin servers.

There are several mechanisms you can use to manage how the WebAccelerator arrives at a TTL value for the compiled responses that it creates. These same mechanisms can also be used to identify content that should not be cached or content that should not be served from cache.

Content Lifetime Settings

There are some values and boundary conditions you can set which control the WebAccelerator's behavior relative to content lifetime. Some of these mechanisms are set using lifetime policies and some of them can be set using the mechanisms described in "Content Lifetime Mechanisms" on page 132.

Maximum Age

A maximum age boundary for cached content can be specified in two ways. First, there is a system-wide maximum age setting. This setting can only be changed by manually editing each Accelerator's configuration file. The system-wide maximum age setting overrides other lifetime mechanisms. No lifetime mechanism can cause the TTL set for a compiled response to be greater than this value. By default, the system-wide maximum TTL value is 24 hours.

In addition to the system-wide maximum age setting, there is a maximum age setting you can specify in a lifetime policy. This setting applies to any HTTP response that matches to the leaf node for which this policy is set. No lifetime mechanism can cause the TTL set for a matching compiled response to exceed this limit.

There is also a system-wide minimum TTL setting. The maximum age that you set for a policy can never be less than this value. If you set the TTL value to something lower than this value, the WebAccelerator ignores it and uses the system-wide minimum TTL setting instead. The default system-wide minimum TTL setting is 2 seconds.

Do not confuse these maximum age settings with the max-age directive used in ESI Surrogate-Control response headers (see "ESI Surrogate-Control Headers" on page 133). The ESI max-age directive is used to define an actual TTL for content. On the other hand, the maximum age settings described here are used to identify boundary conditions beyond which, for example, max-age cannot pass.

It is possible to set maximum age to 0. This forces the WebAccelerator to revalidate all content before serving it from its cache. This revalidation takes the form of issuing a GET-IF-MODIFIED request to your origin servers. If you want to use a maximum age of 0, make sure the system-wide minimum TTL setting is also 0. See the *Administration Guide* for information on changing the `dynamicMinTTL` parameter in `pvsystem.conf`.

Browser Cache Minimum Age

Browser cache minimum age is the minimum time that content can be stored in the web browser's local cache before the browser checks for fresh content from the WebAccelerator. This setting controls the browser's local cache, not the compiled response cache stored on the Accelerators. Once content is downloaded to the browser's local cache, the browser automatically reuses this content without recontacting the WebAccelerator for the period of time identified by the Browser Cache Minimum Age setting. This applies even if the compiled response for that content has been invalidated in the WebAccelerator's cache (see "Invalidating Cached Content" on page 143).

By default this setting is `0` seconds and should be increased only if an acceptable trade off can be made between content freshness and performance. Most browsers still update their cache with fresh content during this period when explicitly requested to do so by the Refresh button. When non-zero, the Browser Cache Minimum Age setting also allows for content to be retained off line for that period of time after being fetched. This setting applies to any HTTP response that matches to a leaf node where the setting is in effect.

Note: If Express Loader is in use, then Browser Cache Minimum Age is ignored for any express loaded objects – the top-level HTML page itself uses the Browser Cache Minimum age value, if set. See "Express Loader Management" on page 84 for more information.

Stand-In Period

The stand-in period identifies how long the WebAccelerator continues to serve content from its cache in the event that both of these are true:

- the content has expired
- your origin servers do not respond when the WebAccelerator proxies to them for fresh content

The stand-in period can be specified in the lifetime policy for a leaf node. If a stand-in period has expired and the WebAccelerator still cannot receive fresh content from your origin servers, the WebAccelerator responds to subsequent requests for that content with an HTTP 404 (not found) response code.

The default stand-in period is 0. If you do not specify a stand-in period, then the WebAccelerator immediately responds with an HTTP 404 (not found) response code if it cannot refresh expired content.

You can also specify a stand-in period using the max-age directive in an ESI Surrogate-Control response header. See "ESI Surrogate-Control Headers" on page 133 for details.

HTTP Lifetime Heuristic

The HTTP Lifetime Heuristic is available only if you are using HTTP headers to identify content lifetime. This heuristic is specified in the lifetime policy for a leaf node. This heuristic has meaning only if you want to use the HTTP LAST_MODIFIED response header to set compiled response TTLs.

For more information on this heuristic, see "HTTP/1.1 Cache-Response Directives" on page 134.

Content Lifetime Mechanisms

There are several mechanisms that you can use to specify content lifetime:

Lifetime Mechanism	Definition
ESI Header	You can use control directives in the ESI Surrogate-control response header to define content TTL values. See "ESI Surrogate-Control Headers" on page 215.
HTTP Headers	You can use HTTP 1.1 lifetime header tags to defined TTL values. See "HTTP/1.1 Cache-Control Headers" on page 133 for more information.
Max Age Settings	The WebAccelerator is configured with a system-wide maximum age for cached content. In addition, you can specify a maximum age setting for content using lifetime policies. See "Defining Lifetime Policies" on page 137 for more information.
Browser Cache Minimum Age	You can specify a minimum age setting for content in the web browser's local cache using lifetime policies. See "Browser Cache Minimum Age" on page 131 for more information.

You use the Policy Editor to identify which lifetime mechanism you are using for cached content. Because you can simultaneously use different lifetime mechanisms, the WebAccelerator obeys a precedence order for these mechanisms. See "Lifetime Mechanism Precedence Rules" on page 138 for more information.

ESI Surrogate-Control Headers

Surrogate-Control headers are sent to the WebAccelerator along with the HTTP response headers. The WebAccelerator receives them as part of the response from your origin servers when the WebAccelerator proxies a request to them. ESI Surrogate-Control headers are used when a site defines page assembly with ESI. See Chapter 9, Assembling Content with ESI for more information.

The ESI Surrogate-Control header directive that you can use to disable caching is no-store. This directive indicates that the WebAccelerator should not cache the response.

The ESI Surrogate-Control header directive that you can use to control content lifetime is max-age. This directive identifies how long the content is to be cached. The value specified by this directive is used by the WebAccelerator as the compiled response TTL value. For details on using and specifying the ESI Surrogate-Control headers, see Appendix C, Response Headers.

If ESI Surrogate-Control headers appear in an HTTP response from your origin servers, the WebAccelerator ignores any HTTP 1.1 Cache-Control headers that also appear. For example, if a response includes an ESI header with the max-age directive and an HTTP 1.1 `no-cache` or `no-store` header, the WebAccelerator caches the response. The exception to this rule is that you can use a lifetime policy to cause the WebAccelerator to ignore the ESI max-age directive. For information on setting lifetime policies, see "Defining Lifetime Policies" on page 137.

HTTP/1.1 Cache-Control Headers

The HTTP 1.1 specification identifies headers that can be used to control web entities such as the WebAccelerator. Cache-Control headers can appear on requests sent by web clients and on responses sent by your origin servers. The Cache-Control general-header field is used to specify directives that must be obeyed by all caching mechanisms along the request/response chain. The directives specify behavior intended to prevent caches from adversely interfering with the request or response. These directives typically override the default caching algorithms. Some directives can only appear in request headers, and some can only appear in response headers. Certain directives can appear in either type of header.

For information, see the HTTP/1.1 specification:
http://www.w3.org/Protocols/rfc2616/rfc2616.html
and specifically the sections that describe Cache-Control headers, sections 13 and 14.

The following sections describes the Cache-Control header directives that the WebAccelerator obeys when they are seen in requests or responses.

HTTP/1.1 Cache-Response Directives

HTTP/1.1 Cache-Response directives that the WebAccelerator obeys can be organized in two groups: directives that cause a response to be uncache-able, and directives that the WebAccelerator uses to help determine compiled response TTLs.

HTTP/1.1 Cache-Response directives that cause the response to be uncache-able are described in the following table. You can tell the WebAccelerator to ignore these no-cache response directives using a setting available in lifetime policies. See "Defining Lifetime Policies" on page 137 for information on how lifetime policies are created.

The HTTP/1.1 Cache-Response directives that the WebAccelerator obeys are:

Directive	**Description**
Expires	The response is uncache-able if the value provided for this directive is in the past. For information on this directive, see section 14.21 of the HTTP 1.1 specification: http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.21
Pragma: no-cache	If the pragma general-header field specifies the no-cache directive, the WebAccelerator does not cache the response. For information on this header, see section 14.32 of the HTTP 1.1 specification: http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.32
Cache-Control: no-cache	If Cache-Control specifies no-cache, the WebAccelerator does not cache the response. For information on the Cache-Control directive, see section 14.9 of the HTTP 1.1 specification: http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.9
Cache-Control: no-store	If Cache-Control specifies no-store, the WebAccelerator does not cache the response.
Cache-Control: max-age	If Cache-Control: max-age=0, the WebAccelerator does not cache the response.
Cache-Control: s-maxage	If Cache-Control: s-maxage=0, the WebAccelerator does not cache the response.
Cache-Control: private	If Cache-Control specifies private, the WebAccelerator does not cache the response.

The HTTP 1.1 response directives that the WebAccelerator uses to determine compiled response TTLs are listed in the following table. Note that the directives are listed by order of priority. For example, if s-maxage and max-age provide two different values, the value provided by s-maxage is used.

Directive	Description
Cache-Control: s-maxage	The TTL used is the current time plus the value specified for s-maxage. Values for this directive are expressed in seconds. For information on the Cache-Control directive, see section 14.9 of the HTTP 1.1 specification: http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.9
Cache-Control: max-age	The TTL used is the current time plus the value specified for max-age. Values for this directive are expressed in seconds.
Expires	The TTL used is the value provided for this directive. Values for this directive are expressed in UTC time. For information on this directive, see section 14.21 of the HTTP 1.1 specification: http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.21
Last-Modified	The WebAccelerator arrives at the TTL using the following formula: `TTL = curr_time + ( curr_time - last_mod_ time ) * last_mod_factor` where: ■ *`curr_time`* is the time that the response is received by the WebAccelerator ■ *`last_mod_ time`* is the time specified by this directive ■ *`last_mod_factor`* is a percentage value used to weight the TTL. It is set using the a lifetime policy. For information on setting lifetime policies, see "Defining Lifetime Policies" on page 137. For information on this directive, see section 14.29 of the HTTP 1.1 specification: http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.29

Freshness Calculation Considerations

The HTTP/1.1 specification indicates that freshness calculations should be based on the date or age value received in the HTTP response. See section 13.2.3 in the HTTP/1.1 specification:
http://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html#sec13.2.3

However, the WebAccelerator currently assumes that these values are 0, so all age calculations are based entirely on the time when the WebAccelerator receives the response.

The extent to which this matters depends on the combination of:

- the time delay between when a response leaves your origin servers and it is received by the WebAccelerator
- the difference between the UTC time set for your origin servers and the UTC time set for the machines hosting the WebAccelerator

In most cases, these issues should be negligible.

HTTP/1.1 Cache-Request Directives

Web clients that are HTTP/1.1 compliant are capable of providing request headers that contain directives which control cache behavior. The WebAccelerator obeys these directives in those situation where they indicate that the web client is unwilling to accept content served from a cache.

If the client sends an HTTP request header that prevents the WebAccelerator from serving the content from its cache, the WebAccelerator proxies the request to your origin servers. This event causes the WebAccelerator to refresh the corresponding content that it has cached, even if that content has not yet expired.

You can tell the WebAccelerator to ignore these Cache-Request directives using a setting available in lifetime policies. See "Defining Lifetime Policies" on page 137 for information on how lifetime policies are created.

The Cache-Request directives that the WebAccelerator obeys are:

Header	Description
Pragma: no-cache	If the pragma general-header field includes the no-cache directive, the WebAccelerator proxies the request to your origin servers. For information on this directive, see section 14.32 of the HTTP/1.1 specification: http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.32
Cache-Control: no-cache	If Cache-Control specifies no-cache, then the WebAccelerator proxies the request to your origin servers. For information on the Cache-Control directive, see section 14.9 of the HTTP/1.1 specification: http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.9
Cache-Control: no-store	If Cache-Control specifies no-store, the WebAccelerator proxies the request.
Cache-Control: max-age	If the value specified on Cache-Control: max-age is less than the current age of the cached content, the WebAccelerator proxies the request.

Header	Description
Cache-Control: min-fresh	The WebAccelerator proxies the request if the following is true: `curr_time + min_fresh_time > content_TTL` where: ■ *`curr_time`* is the time that the request is received by the WebAccelerator ■ *`min_fresh_time`* is the time specified by this directive ■ *`content_TTL`* is the time to live value set for the compiled response that corresponds to the request

Defining Lifetime Policies

You manage lifetime policies using the Policy Editor:

1. Log into the Admin Tool.
2. Go to the Policies screen.
3. Click the Edit link for the policy set whose lifetime policies you want to change.
4. Select the Lifetime link.

For details on the screens you use to define lifetime policies, see the online help.

Lifetime policies allow you to identify the lifetime mechanism that you want to honor for responses matching to the leaf node where the policy is in effect. The lifetime mechanisms that you can use are described in "Content Lifetime Mechanisms" on page 132.

Lifetime policies are only applied to the responses that match their node. They are not relevant to requests that match their node.

If you indicate that you want to use HTTP headers for a lifetime mechanism, then you can also indicate whether you want no-cache headers in the response/request to be honored. See "HTTP/1.1 Cache-Control Headers" on page 133 for information on these headers.

Lifetime policies also allow you to specify age settings that are used to limit the TTL that the WebAccelerator can set for compiled responses. Age settings also allow you to identify a stand-in period and a HTTP Lifetime Heuristic (used only if HTTP headers are selected as a lifetime mechanism). See "Content Lifetime Settings" on page 130 for more information.

Lifetime Policies and the Request Type Hierarchy

Lifetime policies are organized using the Request Type Hierarchy. The Request Type Hierarchy is described in Chapter 5, The Request Type Hierarchy.

Each leaf node in the Request Type Hierarchy has none or one lifetime policy defined for it. This policy has at least one of these defined for it:

- an identification of one or more lifetime mechanisms used for the content that matches to the leaf node. See "Content Lifetime Mechanisms" on page 132.
- age settings for the content that matches to the leaf node. See "Defining Lifetime Policies" on page 137.

Lifetime policies are inherited from a parent node in their entirety. Unlike other types of caching policies where you can add to an inherited policy, lifetime policies only allow you to override the entire policy defined at a parent node.

Lifetime Mechanism Precedence Rules

The TTL set for a compiled response is determined by several potentially conflicting mechanisms. TTL is determined by the system-wide age settings, the settings available in content lifetime values, and the age information that may be supplied using ESI or HTTP mechanisms.

Therefore, the WebAccelerator needs a way to determine what to obey if one or more values conflict. The precedence order used by the WebAccelerator is:

1. Compiled response TTLs cannot be greater than the system-defined maximum lifetime values. Also, compiled response TTLs cannot be shorter than the system-defined minimum lifetime values. See "Maximum Age" on page 130 for more information.
2. No lifetime mechanism can specify a TTL value that is greater than the lifetime policy's maximum age setting, if set.
3. The TTL value indicated by the ESI Surrogate-Control header max-age directive is always obeyed, provided that the ESI max-age value does not violate the boundary conditions described in the previous two bullets, and that ESI headers are turned on in the user interface.
4. The HTTP lifetime headers are used only if the ESI max-age directive is not used or is otherwise not available. Note that in order for HTTP lifetime headers to be obeyed, the leaf node to which the response matches must have a lifetime policy that enables HTTP lifetime headers.

5. If the lifetime policy indicates that no lifetime mechanism is to be used, or if lifetime headers are not otherwise available, then the value set for the lifetime policy's maximum age setting is used for the TTL value.
6. If no lifetime mechanism is used, and the policy does not define a maximum age setting, the TTL value is the same as the system-defined maximum lifetime value.

Lifetime Example

Suppose your site receives the types of requests described in "Request Type Hierarchy Example" on page 60 and your Request Type Hierarchy looks like:

```
-Applications
  -Default
  -Search
-Home
-Images
```

Now, suppose the following:

- Your home page and your image files very rarely change. You are happy to allow the home page and images to be cached for as long as the WebAccelerator defaults allow. If it does change, you use some form of content invalidation to force it to be refreshed. See Chapter 12, Invalidating Cached Content for more information on content invalidation.
- The content served by your general applications changes about once every 4 hours. You expect that you use some form of content invalidation to force a refresh when their content changes, but you want to ensure this content remains in cache for no more than 5 hours without a refresh. Also, to ensure that all content can be subject to invalidation control, you never want to rely solely on the browser's local cache, so there is no minimum time that content can reside in the browser cache before freshness checks.
- Your search application returns data that has wildly varying expiration times. Some of it expires in as short of a time as 10 minutes and some of it lives up to 8 hours. You intend to use the HTTP Expires Cache-Control header to identify the cache time for content served by the search application.
- Finally, because so much of your content overall changes about once every 4 hours or less, as a general policy you are willing to allow the WebAccelerator to serve content that is twice that old, or 8 hours, in the event that your origin servers are not responding to its proxies. You decide to have a stand-in period that is 2 hours longer than your cache period for your search applications because that content changes more rapidly.

Tasks

Do the following for each node in your Request Type Hierarchy:

Node	Application Matching Parameter
`Home`	Set a lifetime policy that indicates a stand-in period of 8 hours, and a maximum lifetime of 24 hours. Provide no other settings for the policy. Your home page is cached for the maximum amount of time allowed by the WebAccelerator.
`Image`	Do the same things as you did for your `Home` node.
`Default`	Create a lifetime policy that sets a maximum cache time of 5 hours. Also set the stand-in period to 8 hours.
`Search`	Create a lifetime policy that turns on HTTP headers support for the lifetime mechanism. Also, set a maximum cache time of 8 hours. Finally set the stand-in period to 10 hours.

You can disallow no-cache HTTP request headers (see "HTTP/1.1 Cache-Request Directives" on page 136). These headers force the WebAccelerator to proxy the request to your origin servers. A browser sends a no-cache request header if the user uses ctrl-refresh (Internet Explorer) or shift-refresh (Netscape). You can use lifetime policies to tell the WebAccelerator to ignore these headers. Doing so may result in a noticeable drop in the traffic experienced by your origin servers, although this is largely determined by your users' browsing habits.

Note: The following procedures walk you through the creation of your policies for the `Default` and `Search` nodes. The procedure you use for setting the `Home` and `Image` node policies is a subset of these procedures.

Actions

Step 1 Setting the Default Node's Lifetime Policy

To create the lifetime policy as described above:

1. Select the `Default` node in the Request Type Hierarchy.
2. Select the Lifetime link.
3. Deselect `ESI Headers` and `HTTP Headers` by unchecking their boxes.
4. In `Age Settings`, set:
   – Browser Cache Minimum Age to `0 Seconds`

   - Maximum Age to `5 Hours`
   - Stand-in Period to `8 Hours`
5. Click the Save button.

You are now done creating this policy. You must publish your policies in order for this change to be used by the WebAccelerator. See "Publishing Policies" on page 45 for more information.

Step 2 Setting the Search Node's Lifetime Policy

To create the lifetime policy as described in "Lifetime Example" on page 139:

1. Select the `Search` node in the Request Type Hierarchy.
2. Make sure the Lifetime link is selected.
3. Deselect `ESI Headers` by unchecking its box.
4. Deselect the boxes for `Ignore no-cache HTTP headers in the request` and `Ignore no-cache HTTP headers in the response`.
5. In `Age Settings`, set:
   - Browser Cache Minimum Age to `0 Seconds`
   - Maximum Age to `8 Hours`
   - Stand-in Period to `10 Hours`
6. Click the Save button.

You are now done creating this policy. You must publish your policies in order for this change to be used by the WebAccelerator. See "Publishing Policies" on page 45 for more information.

Chapter 12

Invalidating Cached Content

Overview of Invalidation
Understanding Invalidation Rules
Manual Invalidation Rules
Raising Invalidation Rules with Triggers
Raising Invalidation Rules with ESI
ESI Invalidation Example
Selector Attribute Reference
Selector Element Reference

Cache invalidation is a powerful tool that allows you to maintain tight coherence between the content currently available on your origin servers and the cached content available in the WebAccelerator. Use cache invalidation whenever you have little tolerance for cached content not being as current as the content on your origin servers.

If content on your site is updated at regular intervals, such as once every day, or once every hour, or even once every minute, you can set lifetime policies to ensure content cached on the WebAccelerator is refreshed with the same frequency. This is different than cache invalidation. Cache invalidation works by allowing you to expire cached content before it has reached its time to live (TTL) value. This method works well when updates to content are event-driven, such as when another item is added to a shopping cart or a request contains a new auction bid.

There are several types of cache invalidation:

- manual invalidation, where you decide that cached content is no longer current, (perhaps you just updated your site) and you specify the content to be expired
- triggered invalidation, where you create rules that describe a triggering request and specify the content to be expired. Whenever a request that matches the trigger is received, that content is expired.
- ESI invalidation, where you create an XML file that describes and initiates the invalidation and is sent to the WebAccelerator by your origin servers, possibly whenever a database update occurs

Cache invalidation allows you to describe the events that should force the refresh of cached content.

Overview of Invalidation

When the WebAccelerator receives an HTTP request, it must decide whether it can service that request from cache. One of the things that prevents the WebAccelerator from servicing the request from cache is if the cached content needed to serve the request is expired. If a request matches to an invalidation rule, the WebAccelerator treats the request as if any cached content had expired. It proxies the request to the origin servers, receives their response, sends the response to the client, and also caches it, refreshing any existing cached content.

There are three types of invalidation rules you can create, using three different mechanisms:

- manual invalidation

  You manually submit an invalidation rule from the Applications screen. See "Manual Invalidation Rules" on page 146 for more information.

- invalidation triggers

  You create an invalidation rule and define a triggering request. The rule is triggered based on elements seen on an HTTP request. See "Raising Invalidation Rules with Triggers" on page 147 for more information.

- ESI cache invalidation

  You create an XML file that contains an invalidation rule that is sent to the WebAccelerator by your origin servers. See "Raising Invalidation Rules with ESI" on page 152 for more information.

Understanding Invalidation Rules

Invalidation rules identify requests that must be proxied to your origin servers. You specify the elements on a request that result in invalidation. When a request is received that matches the elements you specify, the cached content that normally would have been used to service the request is invalidated, because the request is proxied and the content replaced with the response to the proxy.

Usually there is a range of requests that match an invalidation rule, so there is a corresponding range of compiled responses that are invalidated. For example, suppose you wanted to invalidate any content stored in the WebAccelerator that is related to the application:

```
/apps/doSomething.jsp
```

You raise an invalidation rule based on that path. Any request matching that path is invalidated, that is, it is proxied to your origin servers.

Invalidation rules can be quite broad. For example, a rule can invalidate any request containing a cookie from a specified set of cookies. Depending on how your site is designed, this can invalidate a significant percentage of the compiled responses you currently have cached. You can even invalidate the entire contents of the cache with a single rule by specifying any request received by your domain.

Use care when specifying your invalidation rule. Although there are times when you need to invalidate a significant portion of the cache, this can put a large load on your origin servers as they attempt to respond to the many proxied requests. For this reason, try to specify your invalidation rules as narrowly as possible, while still ensuring that any content which should be replaced is targeted by the invalidation.

When Invalidation Rules are in Effect

For a raised invalidation rule to be in effect for a request, three things must be true:

1. The rule must have reached its effective time.

   You can specify invalidation rules to be effective immediately, or you can set an effective date that must be reached before the rule goes into effect.

2. The information that appears on the request must match all the HTTP Request Data Type Parameters specified for the invalidation rule.

   For example, the invalidation rule might have two parameters. One parameter specifies that the `product` query parameter must be set to `Computers`. The other parameter specifies that the request must be for any path beginning with `/apps`. In this case, these requests are candidates for invalidation:

   ```
   http://www.somesite.com/apps/shop.jsp?action=show&product=Computers
   http://web1.somesite.com/apps/search/simple.jsp?product=Computers&
   category=desktop
   ```

   while these are not:

   ```
   http://www.somesite.com/shop.jsp?action=show&product=Computers
   http://web1.somesite.com/apps/search/simple.jsp?product=Organizers&
   category=desktop
   ```

3. The cached content corresponding to the request must have a last refresh time that is earlier than the effective date on the invalidation rule.

   This ensures that a compiled response is not invalidated more than once under the same invalidation rule. A compiled response's refresh time identifies the last time the WebAccelerator refreshed that compiled response with content from your origin servers. If the compiled response was refreshed after the invalidation rule went into effect, it is considered to be current and does not need to be refreshed again.

Invalidation Rule Lifetimes

Invalidation rules usually are targeted at a range of compiled responses. No compiled response is invalidated until a request that matches the rule is received for that compiled response. A range of compiled responses is not invalidated until enough varying requests have been received that finally each compiled response in the range was identified by a request and invalidated and refreshed. This can take some time after the invalidation rule is raised. The WebAccelerator ensures that every targeted compiled response is refreshed before it discards the rule by setting an appropriate lifetime for the rule.

The WebAccelerator sets the lifetime by examining the maximum lifetime values set for all compiled responses targeted by the rule and using the largest value it finds as the lifetime for the rule. Once the longest compiled response lifetime has been reached, those compiled responses would be considered expired and must be refreshed, even if they have not yet been invalidated.

For example, suppose you create an invalidation rule for any requests that have either `/apps` or `/srch` in their path. Your Request Type Hierarchy includes two nodes with application matching rules set so that requests with `/apps` in the path match to one node and requests with `/srch` match to the other node. The `/srch` node has a lifetime policy specifying a maximum lifetime of 15 minutes, while the `/apps` node has no lifetime policy.

Compiled responses created to service requests matching the `/srch` node have a maximum lifetime of 15 minutes. Compiled responses created to service requests matching the `/apps` node use the system maximum lifetime, which is 24 hours. This means the WebAccelerator assigns a lifetime to your invalidation rule of 24 hours.

During the 24 hours the rule is in effect, matching compiled responses are either refreshed due to the invalidation rule, or they are refreshed because they have exceeded their individual TTL settings. Either way, any content matching the invalidation rule's parameters is refreshed at least once before the WebAccelerator discards the rule.

For information on setting lifetime policies, see Chapter 11, Cache Control and Content Lifetime.

Manual Invalidation Rules

The WebAccelerator provides a mechanism in the Policy Editor that allows you to manually trigger an invalidation rule. This mechanism is not a policy in that it does not have to be published. Instead, it allows you to define an invalidation rule which you can then activate by clicking a button in the user interface.

Configuring Manual Invalidation

You configure manual invalidation by using the Admin Tool. Manual invalidation can only be performed on active applications. If the WebAccelerator is not currently accelerating that application, there is no cache associated with the application to invalidate.

Go to Applications in the Admin Tool. Select the Invalidate Content link. For details on the screens you use to define manual invalidation, see the online help.

Manual Invalidation Example

Suppose that your car rental site serves groups of images that use this naming convention:

```
/images/automobile_228a9.gif
/images/automobile_q5df90.gif
/images/automobile_nnp8g5.gif
```

Because these images are stable, you have a very long cache TTL set for them. However, you recently updated all your automobile_*xxx* images and you want those updated images to be refreshed in the WebAccelerator cache.

To create this manual invalidation:

1. Log into the Admin Tool.
2. Go to the Applications screen.
3. Click the Invalidate Content link.
4. Select `Invalidate content that matches the following URI expression`.
5. In the text window, enter the appropriate regular expression:
   `/\/images\/automobile_.*\.gif/`
6. In the applications section, check the box selecting your car rental application.
7. Click the Invalidate button.

The invalidation object has now been raised. The next time each one of your automobile gifs are requested from your site, you are guaranteed that the image is served from your origin servers, and that the corresponding compiled responses in your WebAccelerator are refreshed.

Raising Invalidation Rules with Triggers

Triggered invalidation rules are a caching policy that allow you to trigger an invalidation when a request that matches the trigger specifications is received.

Triggered invalidation rules are processed before any response matching, and so only apply to requests that match to their node, never to responses. When specifying a triggered invalidation rule, you are identifying two types of requests:

- requests that trigger the invalidation

  Whenever a request that matches the source specification is received, the invalidation rule is triggered.

- requests that map to compiled responses which should be refreshed

  Whenever a request that matches the target specification is received (after a triggering request has been received), that request is proxied and the compiled response that would have been used to service the request is refreshed.

For example, suppose you are running an auction site. Requests that enter a bid for an auction use the form:
`http://www.yourauctions.com/bid.jsp?auction=3910387&bid=24.50`

Upon receiving a bid, you want to invalidate any pages that display the current bid for that auction. For your site, requests that retrieve current bids use the form:
`http://www.yourauction.com/show.jsp?auction=3910387`

A triggered invalidation allows you to define an invalidation that is triggered if a request for application `bid.jsp` is received. The invalidation rule that is raised forces a proxy for any request whose:

- path ends with `show.jsp`
- `auction` query parameter is set to the same value as the `auction` query parameter on the triggering request

Defining Invalidation Trigger Policies

You manage invalidation trigger policies using the Policy Editor:

1. Log into the Admin Tool.
2. Go to the Policies screen.
3. Click the Edit link for the policy set whose invalidation trigger policies you want to change.
4. Select the leaf node in the Request Type Hierarchy that describes the types of requests that should trigger the invalidation. Invalidation trigger policies are always created on the node that the triggering requests match to, not the node that the invalidation targets match. In other words, the requests that match to this node should be the requests that activate the policy, not the requests whose compiled responses get invalidated.

   You must select a node before you can access the Invalidations screen.
5. Select the Invalidations link.

For details on the screens you use to define invalidation trigger policies, see the online help.

Sources for Triggers

The source of a triggered invalidation rule is the request received by the WebAccelerator which activates, or triggers, the invalidation rule. The request must match to the leaf node for which the invalidations policy is defined. However, not all requests which match this node must trigger the invalidation. You can define a subset of matching requests by specifying additional parameters for the trigger source as part of the policy. Then only this subset of requests trigger the invalidation.

For example, assume that to match to a particular node, a request must include `/apps/shopping` in the path and the query parameter `cart` must be present. For the source of the invalidations policy on this node, you can specify that a request must include the query parameter `cart` and the value of `cart` must equal `Add`. In this case, not all requests that match to the node trigger the invalidation. Only requests where `cart=add` trigger the invalidation.

You specify the source for an invalidation trigger using HTTP Request Data Type Parameters, similar to many other policies. You specify the parameter and the value needed for a match. An invalidation rule is triggered if an HTTP request matches to the leaf node and it matches all the specified source parameters.

Targets for Triggers

The targets of a triggered invalidation are all the requests which you want to be proxied. The result of the proxies is that cached content that normally would have serviced the requests is refreshed. You define the target requests by specifying the HTTP Request Data Type Parameters and their values that are required for a match. If a match occurs, the request is proxied.

Incoming requests are not examined and compared to the target specifications until after a request that matches the trigger source specification ("Sources for Triggers") is received. After that point, whether a potential target request is actually invalidated depends on whether the request satisfies all the conditions described in "When Invalidation Rules are in Effect" on page 145.

When specifying a target, you must provide at least one parameter based on the Path data type. Remember, any request is a candidate for the invalidation target, not just requests that matched to the node the policy is on. It is only the source requests that had to be matched to this node first.

Specifying a Target

When you specify what your target requests should look like, what values their elements should match to, you can specify the value to match or you can specify that the value should match a value found in the source request that triggered the invalidation rule.

For example, you can define a target parameter that must be identical to that of the `action` query parameter on the source request. If `action` on the source request was set to `bid`, then the target parameter matches any request whose `action` parameter is also set to `bid`. However, the parameter you specify as a source for the value does not have to be the same parameter you are specifying for the target. For example, you can create a target parameter such that the value set for the `userID` query parameter must match the value set for the source's `sessionID` cookie. If the request that triggered the invalidation had:

```
COOKIE: sessionID=JBabsen
```

then a request with this query parameter:

```
userID=JBabsen
```

matches the invalidation target parameter.

Invalidation Triggers and the Request Type Hierarchy

Invalidation trigger policies are organized using the Request Type Hierarchy. Request Type Hierarchy inheritance is not allowed for invalidation trigger policies. Invalidation trigger policies can be defined only on leaf nodes in the hierarchy.

Each leaf node in the Request Type Hierarchy can have multiple invalidation triggers defined for it. Each such policy has at least one source parameter of any type and one target Path parameter defined.

Invalidation Trigger Example

Suppose your site receives the types of requests described in "Request Type Hierarchy Example" on page 60 and your Request Type Hierarchy looks like:

```
-Applications
  -Default
  -Search
-Home
-Images
```

Suppose that any request for `/apps/doSomething.jsp` changes content returned by `/srch/doSimpleSearch.jsp`. However, requests for `doSimpleSearch.jsp` should be updated only if its `producttype` query parameter matches the value set for the `group` query parameter seen on `doSomething.jsp`.

That is, if the WebAccelerator receives this request:
`http://www.somesite.com/apps/doSomething.jsp?group=Doors&....`

then it should proxy any subsequent requests that look like:
`http://www.somesite.com/srch/doSimpleSearch.jsp?producttype=Doors&....`

You want to create an invalidation trigger to handle these updates. You want to create the invalidations policy on the Default node, because you know that all requests for `/apps/doSomething.jsp` match to that node. Its target, requests for `doSimpleSearch.jsp`, always match to the Search node and should include a `producttype` query parameter whose value matches the value of group in the triggering request for `/apps/doSomething.jsp`.

To create this invalidation trigger:

1. Select the `Default` node in your Request Type Hierarchy.
2. Select the Invalidations link and click the Add button.
3. Enter a meaningful description for the trigger, such as `doSomething invalidated producttype`.
4. In the Source table, select `Path` from the Add Parameter drop-down list and click the Add button.
5. In the Add Source Parameter screen, add the `doSomething.jsp` application path to the Values box. The result is that this trigger raises only for requests for this application. All other requests that match to the `Default` node do not raise the trigger.
6. When you are done, click the Save button. The definition of the trigger's source now appears in the Source table.
7. In the Target table, select `Path` from the Add Parameter drop-down list and click the Add button.
8. Select Value Group from the Type drop-down list and enter the path for `/srch/doSimpleSearch.jsp`. Click the Save button.
9. Select Query Parameter from the parameter list and click Add.
10. Enter the name of the target query parameter, `producttype`.
11. From the dropdown list, select Query Parameter from Source and enter the name of the source query parameter, `group`.
12. Click the OK button.
13. You now see the new rule in the Invalidation Triggers summary table. Check the Active checkbox.
14. Click the Save button.

You are now done creating the trigger. In order for the new policy to be in effect for your site, it must be published. See "Publishing Policies" on page 45 for more information.

Raising Invalidation Rules with ESI

ESI is an open specification that identifies mechanisms for controlling web surrogates such as the WebAccelerator. Within the context of cache invalidation, ESI offers the ESI Cache Invalidation Specification. It can be found here:
http://www.esi.org/invalidation_protocol_1-0.html

The WebAccelerator supports this specification, with some caveats. The WebAccelerator also provides several extensions to this specification.

Sending ESI Invalidation Requests

You can raise an invalidation rule in the WebAccelerator by sending an ESI invalidation request to the WebAccelerator. This request is sent to the same host and port that you use to connect to the Admin Tool.

The invalidation request must:

- use the HTTP 1.1 protocol
- correctly identify the content length of the invalidation request using the Content-Length request header. In addition, the size of the request must be less than 50KB (50,000 bytes).
- use the POST method
- use the Authorization request header to specify a basic authorization
- request this URI:
  `/servlet/COM.pivia.esi.server.InvalidationServlet`

For example:

```
POST /servlet/COM.pivia.esi.server.InvalidationServlet HTTP/1.1
Authorization: Basic cWExOnOhc3N3z3Jk
Content-Length: 328

invalidation request goes here
    ...
```

For information on the portion of the example request denoted by `invalidation request goes here`, see "Creating an ESI Invalidation Request" on page 154.

Authorizing an Invalidation Request

Every ESI invalidation request must include an authorization string that provides the name of the application profile for the invalidation and the invalidation service password. This string must be sent to the WebAccelerator using the HTTP Authorization request header. You must indicate basic authorization on the header and you must base64-encode the authorization string:

```
Authorization: Basic cWExOnOhc3N3z3Jk
```

If seen in clear-text, the authorization string must be formatted as:

`application``:`*`password`*

where:

`application`	is the name of the application profile, and is the same name you see on the Applications screen.
`password`	is the password set for the invalidation service account. You can change the password for the invalidation service account from the main Applications screen. The default password set for the account when the WebAccelerator was installed is `invalidator`.

Note: ESI invalidations can only invalidate requests for domains that appear on the host map for the application you specify. On the Applications screen, you can see a list of the domains for each application.

Invalidation Response

When the WebAccelerator receives and processes an ESI invalidation request, it responds with an 200 response code. This response is sent as soon as the WebAccelerator creates the invalidation rule.

The invalidation response provides information that identifies the invalidation request it processed, the invalidation objects that were sent with the request, and the results of processing that objects. For example:

```
1:  200
2:  OK
3:  Content-Type: text/xml
4:  Content-Length: 422
5:  Servlet-Engine: Tomcat Web Server/3.2.1 (JSP 1.1; Servlet 2.2; Java
6:  1.3.0_02; Linux 2.2.17-14smp i386; java.vendor=Sun Microsystems Inc.)
7:
8:  <?xml version="1.0"?>
9:  <!DOCTYPE INVALIDATIONRESULT SYSTEM "invalidation.dtd">
10: <INVALIDATIONRESULT VERSION="WCS-1.0">
11: <OBJECT_RESULT>
12: <OBJECT BATCH='1083' SEQUENCE='2'>
13: <ADVANCEDSELECTOR URIPREFIX='/apps/' URIEXP='/apps/index.html'>
14: <COOKIE NAME='username' VALUE='pwang'></COOKIE>
15: </ADVANCEDSELECTOR>
16: <ACTION REMOVALTTL='0'></ACTION>
17: </OBJECT>
18: <RESULT ID="1" STATUS="SUCCESS" NUMINV="1"/>
19: </OBJECTRESULT>
20: </INVALDIATIONRESULT>
```

Lines 1 - 6 are the invalidation response headers that you receive for a successful invalidation request.

Lines 8 - 20 are the response body. Lines 8 and 9 identify the body as a valid XML 1.0 document that uses the invalidation DTD.

Lines 10 - 20 identify the result of the invalidation. Lines 11 - 19 identify the result for a specific invalidation object. There is one invalidation object result for each invalidation object that you sent in your request. Lines 12 - 17 identify the actual invalidation object that was processed by the WebAccelerator.

Notice that line 18 gives the result status for the invalidation object. The ESI invalidation specification indicates that this result should identify the number of cached objects invalidated by the invalidation object. This number should appear in the NUMINV field. The WebAccelerator deviates from the ESI invalidation specification in that it always indicates NUMINV is 1.

Creating an ESI Invalidation Request

An ESI invalidation request is a valid XML document that identifies one or more invalidation objects. The invalidation request is sent to the WebAccelerator using the HTTP 1.1 protocol. See "Sending ESI Invalidation Requests" on page 152 for more information.

An ESI invalidation request has the following format:

```
1:  <?xml version="1.0" ?>
2:  <!DOCTYPE INVALIDATION SYSTEM "invalidation.dtd">
3:  <INVALIDATION VERSION="WCS-1.0">
4:  <OBJECT>
5:     ....
6:  </OBJECT>
7:  <OBJECT>
8:     ....
9:  </OBJECT>
10: <OBJECT>
11:    ....
12: </OBJECT>
13: ...
14: </INVALIDATION>
```

Lines 1-3 and 14 are required. They must be used in every invalidation request exactly as they appear here.

Lines 4-6, 7-9, and 10-12 represent invalidation objects. There must be at least one invalidation object included on the request. There can be any number of invalidation objects identified on the request so long as the size of the request is less than 50 KB.

The WebAccelerator raises one invalidation rule for each valid invalidation object. Each object definition provides an invalidation selector (either basic or advanced) and an invalidation action (see "Invalidation Action" on page 158).

Basic Invalidation Selectors

<BASICSELECTOR URI="*absolute_path*" />

Basic selectors identify a URI that the WebAccelerator uses as a target parameter for the invalidation rule. Any request that matches the URI identified by the basic selector is a candidate for invalidation. Whether the request is in fact invalidated depends on whether the request satisfies all the conditions described in "When Invalidation Rules are in Effect" on page 145.

Note that you can only invalidate objects for which you have invalidation rights. See "Authorizing an Invalidation Request" on page 153 for details.

This example of an invalidation request uses a basic selector. This invalidation request causes the WebAccelerator to raise an invalidation rule based on the path: /apps/bid.jsp

Any HTTP request the WebAccelerator receives in which the path /apps/bid.jsp appears is a candidate for invalidation.

```
POST /servlet/COM.pivia.esi.server.InvalidationServlet HTTP/1.1
HOST: www.somesite.com
Authorization: Basic aW52YWxpZGF0aW9uOnBhc3N3b3Jk\012
Content-Length: 191
```

```
<?xml version="1.0" ?>
<!DOCTYPE INVALIDATION SYSTEM "invalidation.dtd">
<INVALIDATION VERSION="WCS-1.0">
<OBJECT>
<BASICSELECTOR URI="/apps/doSomething.jsp" />
<ACTION />
</OBJECT>
</INVALIDATION>
```

Advanced Invalidation Selectors

```
<ADVANCEDSELECTOR URIPREFIX="prefix" [URIEXP="regular expression"]
[HOST="host"]
      [METHOD="method"] [BODYEXP="regular expression"] >
      optional selector element
     optional selector element
      ...
</ADVANCEDSELECTOR>
```

Advanced selector tags provide a flexible mechanism for identifying content that the WebAccelerator must invalidate. You use a combination of ADVANCEDSELECTOR tag attributes and selector elements to identify content that you want invalidated. Note that a request must match all the criteria identified by the ADVANCEDSELECTOR tag attributes and selector elements in order for it to be invalidated.

The ADVANCEDSELECTOR attributes that you can use are defined as:

Attribute	Description
URIPREFIX	Required attribute that you use to identify the path used in an HTTP request.
HOST	Optional attribute that you use to identify a literal value for the HTTP HOST request header.
METHOD	Optional attribute that you use to identify a literal value for the HTTP method.
BODYEXP	Optional attribute that you use to identify a regular expression that is matched to the body of an HTTP request.
URIEXP	Optional attribute that you use to identify a regular expression that is matched to the path used in an HTTP request.

The selector elements that you can use are:

Element	Description
COOKIE	Optional element used to identify a cookie and its value that must appear in the request.
HEADER	Optional element used to identify an HTTP/1.1 header and its value that must appear in the request.
OTHER TYPE PV_QUERY_EXP	Optional element used to identify a query parameter and its value that must appear in the request. A regular expression is used to identify both the query parameter name and the parameter value.
OTHER TYPE PV_UNAME_QUERY_EXP	Optional element used to identify an unnamed query parameter and its value that must appear in the request. A regular expression is used to identify the parameter value.
OTHER TYPE PV_COOKIE_EXP	Optional element used to identify a cookie and its value that must appear in the request. A regular expression is used to identify the cookie value.
OTHER TYPE PV_USER_AGENT_EXP	Optional element used to identify a USER_AGENT value that must appear in the request. A regular expression is used to identify the USER_AGENT value.
OTHER TYPE PV_REFERER_EXP	Optional element used to identify a REFERER value that must appear in the request. A regular expression is used to identify the REFERER value.
OTHER TYPE PV_HEADER_EXP	Optional element used to identify an HTTP/1.1 header and its value that must appear in the request. A regular expression is used to identify the header value.
OTHER TYPE PV_CUSTOM_HEADER_EXP	Optional element used to identify a structured header together with one of its parameters and the value for that parameter that must appear in the request. A regular expression is used to identify the parameter value.
OTHER TYPE PV_HOST_ALT	Optional element used to identify an HTTP HOST request header value that must appear in the request.
OTHER TYPE PV_SEGMENT_L_R_EXP	Optional element used to identify a path segment or a segment parameter and its value that must appear in the request. A regular expression is used to identify the segment value. Ordinals are identified by counting from left to right.

Element	Description
OTHER TYPE PV_SEGMENT_R_L_EXP	Optional element used to identify a path segment or a segment parameter and its value that must appear in the request. A regular expression is used to identify the segment value. Ordinals are identified by counting from right to left.
OTHER TYPE PV_EXT_EXP	Optional element used to identify an extension that must appear in the request. A regular expression is used to identify the extension.

Invalidation Action

`<ACTION [REMOVALTTL=”`*`seconds`*`”]|[PV_ABS_REMOVALTTL] [PV_NO_CACHE_PERIOD]/>`

Identifies the effective time of the invalidation rule that the WebAccelerator raises as a result of the invalidation object.

`REMOVALTTL` is a non-negative time in seconds that identifies the time when the invalidation rule becomes effective. The effective time is the time that the WebAccelerator receives the invalidation request plus the `REMOVALTTL`. If `REMOVALTTL` and `PV_ABS_REMOVALTTL` are both provided, `REMOVALTTL` is used.

`PV_ABS_REMOVALTTL` is a non-negative time in seconds that identifies the absolute time when the invalidation rule becomes effective. The value provided here is the number of seconds since the Unix Epoch (January 1, 1970). `PV_ABS_REMOVALTTL` is a Swan Labs extension to the ESI Invalidation protocol.

If `REMOVALTTL` or `PV_ABS_REMOVALTTL` is not specified, then the invalidation rule’s effective time is when the WebAccelerator receives the invalidation request.

`PV_NO_CACHE_PERIOD` is a non-negative time in seconds that identifies a delay for the invalidation rule’s effective time. During this delay, the WebAccelerator proxies all requests that match the invalidation rule. When the delay has expired, the WebAccelerator honors the invalidation rule as normal. See “When Invalidation Rules are in Effect” on page 145.

For example:

```
<?xml version="1.0" ?>
<!DOCTYPE INVALIDATION SYSTEM "invalidation.dtd">
<INVALIDATION VERSION="WCS-1.0">
<OBJECT>
    <BASICSELECTOR URI=”/apps/bids.jsp” />
    <ACTION REMOVALTTL=”900”/>
</OBJECT>
<OBJECT>
    <ADVANCEDSELECTOR URIPREFIX=”/apps/” >
        <COOKIE NAME=”PLAN” VALUE=”PLATINUM” />
```

```
11:     </ADVANCEDSELECTOR>
12:     <ACTION />
13: </OBJECT>
14: </INVALIDATION>
```

This invalidation request identifies two invalidation objects. The invalidation rule that is raised for the object defined in lines 4 - 7 is effective 15 minutes after the invalidation request is received by the WebAccelerator. The invalidation rule raised for the object defined in lines 8 - 13 is effective immediately.

A given request can match both invalidation rules:

```
GET /apps/bids.jps HTTP/1.1
HOST: www.somesite.com
COOKIE: PLAN=PLATINUM
```

If this situation arises, then a request can be invalidated once for each object. The first invalidation occurs immediately (this satisfies the invalidation object defined on lines 8 - 12) and the second invalidation occurs in 15 minutes.

However, a request that matches both invalidations can be invalidated only once. This happens if the request is seen very infrequently. If the request first appears 20 minutes after the above invalidation request arrives in the WebAccelerator, then the request is invalidated just once. That invalidation satisfies both invalidation objects.

ESI Invalidation Example

Suppose you have an application that returns a web page that uses five different images. Every time the web page is updated, the images are also updated. Therefore, when you invalidate the cached content for the application, you also have to invalidate the images.

The application is queried as follows:

```
http://www.somesite.com/apps/magic.jsp?....
```

The image files that this application uses are as follows:

```
http://www.somesite.com/images/magic_1.jpg
http://www.somesite.com/images/magic_2.jpg
http://www.somesite.com/images/dinabank.jpg
http://www.somesite.com/images/masthead.jpg
http://www.somesite.com/images/featuredImage.jpg
```

In this case, you need to create 5 invalidation objects. One of these objects invalidates the first two image URLs using a regular expression. You can invalidate all of these with a single invalidation request:

```
POST /servlet/COM.pivia.esi.server.InvalidationServlet HTTP/1.1
Authorization: Basic cWExOnOhc3N3z3Jk
Content-Length: 617
```

```
<?xml version="1.0" ?>
<!DOCTYPE INVALIDATION SYSTEM "invalidation.dtd">
<INVALIDATION VERSION="WCS-1.0">
<OBJECT>
      <BASICSELECTOR URI="/apps/magic.jsp" />
      <ACTION />
</OBJECT>
<OBJECT>
      <ADVANCEDSELECTOR URIPREFIX="/images/" >
      <OTHER TYPE="PV_SEGMENT_L_R_EXP" NAME="2" VALUE="magic.*\.jpg" />
      </ADVANCEDSELECTOR>
      <ACTION />
</OBJECT>
<OBJECT>
      <BASICSELECTOR URI="/images/dinabank.jpg" />
      <ACTION />
</OBJECT>
<OBJECT>
      <BASICSELECTOR URI="/images/masthead.jpg" />
      <ACTION />
</OBJECT>
<OBJECT>
      <BASICSELECTOR URI="/images/featuredImage.jpg" />
      <ACTION />
</OBJECT>
</INVALIDATION>
```

You send this request to the host and port that is configured to accept ESI invalidation requests for your site. This should be the same host and port that you use to access the Admin Tool, for example:

```
https://mynwa.mydomain.com:8443
```

The authorization that you use must be a valid ESI authorization string (that is base64-encoded), and it must be an authorization that has invalidation rights for the `www.somesite.com` domain.

Once the invalidation rule is raised, the next instance of any of the requests described in this example is guaranteed to be proxied to your origin servers.

Selector Attribute Reference

The URIPREFIX attribute is required. All other selector attributes are optional.

URIPREFIX

```
<ADVANCEDSELECTOR URIPREFIX=”prefix” >
      ...
</ADVANCEDSELECTOR>
```

This ADVANCEDSELECTOR attribute is required.

URIPREFIX identifies a prefix that an HTTP request must either begin with or match exactly in order for the request to be a candidate for invalidation. Because this attribute accepts a prefix, the value provided here must begin and end with a slash (/).

For example, if an invalidation object is defined using:

```
<OBJECT>
<ADVANCEDSELECTOR URIPREFIX=”/apps/srch/”>
</ADVANCEDSELECTOR>
<ACTION />
</OBJECT>
```

then this HTTP request is a candidate for invalidation:

```
http://www.somesite.com/apps/srch/simpleSearch.jsp?....
```

while this is not:

```
http://www.somesite.com/apps/doSomething.jsp?....
http://www.somesite.com/apps?....
```

HOST

```
<ADVANCEDSELECTOR URIPREFIX=”prefix” HOST=”hostname”>
      ...
</ADVANCEDSELECTOR>
```

The optional HOST attribute provides a literal value to which the HTTP HOST request header must be set in order for the WebAccelerator to consider the request as a candidate for invalidation.

HOST is used with the required URIPREFIX attribute. An HTTP request must match the value provided on the URIPREFIX attribute before the WebAccelerator examines the HOST request header against the value provided here.

For example, if an invalidation object is defined using:

```
<OBJECT>
<ADVANCEDSELECTOR URIPREFIX=”/apps/” HOST=”www.somesite.com”>
</ADVANCEDSELECTOR>
```

```
<ACTION />
</OBJECT>
```

then this HTTP request is a candidate for invalidation:

```
GET /apps/bid.jsp HTTP/1.1
HOST: www.somesite.com
```

while this is not:

```
GET /apps/bid.jsp HTTP/1.1
HOST: www.adifferentsite.com
```

METHOD

```
<ADVANCEDSELECTOR URIPREFIX="prefix" METHOD="method">
      ...
</ADVANCEDSELECTOR>
```

The optional `METHOD` attribute provides a literal value to which the HTTP method must be set in order for the WebAccelerator to consider the request as a candidate for invalidation.

`METHOD` is used in coordination with the required `URIPREFIX` attribute. An HTTP request must match the value provided on the `URIPREFIX` attribute before the WebAccelerator examines the request's method against the value provided here.

For example, if an invalidation object is defined using:

```
<OBJECT>
<ADVANCEDSELECTOR URIPREFIX="/apps/" METHOD="POST">
</ADVANCEDSELECTOR>
<ACTION />
</OBJECT>
```

then this HTTP request is a candidate for invalidation:

```
POST /apps/bid.jsp HTTP/1.1
```

BODYEXP

```
<ADVANCEDSELECTOR URIPREFIX="prefix" BODYEXP="regular expression">
      ...
</ADVANCEDSELECTOR>
```

The optional `BODYEXP` attribute provides a regular expression. The body of the HTTP request must match the regular expression given here in order for the HTTP request to be a candidate for invalidation.

This attribute is meaningful only if the request uses the POST method.

`BODYEXP` is used in coordination with the required `URIPREFIX` attribute. An HTTP request must match the value provided on the `URIPREFIX` attribute before the

WebAccelerator examines the request against the regular expression provided here. Regular expression support is described in Appendix D, Regular Expressions.

For example, if an invalidation object is defined using:

```
<OBJECT>
<ADVANCEDSELECTOR URIPREFIX="/apps/" BODYEXP=".*username=jbabsen.*">
</ADVANCEDSELECTOR>
<ACTION />
</OBJECT>
```

then this HTTP request is a candidate for invalidation:

```
POST /apps/bid.jsp HTTP/1.1
HOST: www.somesite.com

familiarname=Jen&surname=Babsen&username=jbabsen&
accountnumber=R65MM90c32
```

URIEXP

Caution: For performance reasons, this attribute should be avoided. Use `URIPREFIX` instead.

```
<ADVANCEDSELECTOR URIPREFIX="prefix" URIEXP="regular expression">
      ...
</ADVANCEDSELECTOR>
```

The optional `URIEXP` attribute provides a regular expression. The full path used by an HTTP request must match the regular expression given here in order for the HTTP request to be a candidate for invalidation.

`URIEXP` is used with the required `URIPREFIX` attribute. An HTTP request must match the value provided on the `URIPREFIX` attribute before the WebAccelerator examines the request against the regular expression provided here. Regular expression support is described in Appendix D, Regular Expressions.

For example, if an invalidation object is defined using:

```
<OBJECT>
<ADVANCEDSELECTOR URIPREFIX="/apps/" URIEXP=".*\.jsp$">
</ADVANCEDSELECTOR>
<ACTION />
</OBJECT>
```

then this HTTP request is a candidate for invalidation:

```
http://www.somesite.com/apps/srch/simpleSearch.jsp?....
```

while this is not:

```
http://www.somesite.com/apps/srch/include.html?....
```

Selector Element Reference

All selector elements are optional.

COOKIE

```
<ADVANCEDSELECTOR URIPREFIX="prefix">
      <COOKIE NAME="name" VALUE="value" />
      <COOKIE NAME="name" VALUE="value" />
      <COOKIE NAME="name" VALUE="value" />
      ...
</ADVANCEDSELECTOR>
```

An ADVANCESELECTOR can have 0 or more COOKIE elements. The HTTP request must have all of the cookies identified here, and those cookies be set to the value identified for them, in order for the HTTP request to be a candidate for invalidation.

The NAME attribute identifies the name of a cookie. The value provided here must be a literal.

The VALUE attribute identifies the value to which the named cookie must be set. The value provided here must be a literal.

This element is used in coordination with the attributes provided for ADVANCEDSELECTOR. An HTTP request must match all the conditions identified by the ADVANCEDSELECTOR attributes before the WebAccelerator examines the request against the COOKIE elements.

For example, if an invalidation object is defined using:

```
<OBJECT>
<ADVANCEDSELECTOR URIPREFIX="/apps/">
<COOKIE NAME="PLAN" VALUE="GOLD" />
<COOKIE NAME="GROUP" VALUE="ADMINISTRATOR" />
</ADVANCEDSELECTOR>
<ACTION />
</OBJECT>
```

then this HTTP request is a candidate for invalidation:

```
GET /apps/showValue.jsp HTTP/1.1
HOST: www.somesite.com
COOKIE: PLAN=GOLD;GROUP=ADMINISTRATOR
```

HEADER

```
<ADVANCEDSELECTOR URIPREFIX="prefix">
      <HEADER NAME="name" VALUE="value" />
      <HEADER NAME="name" VALUE="value" />
      <HEADER NAME="name" VALUE="value" />
      ...
</ADVANCEDSELECTOR>
```

An ADVANCESELECTOR can have 0 or more HEADER elements. Each HEADER element identifies an HTTP/1.1 header and its value. The HTTP request must have all of the headers identified here, and those headers be set to the value identified for them, in order for the HTTP request to be a candidate for invalidation.

The NAME attribute identifies the name of a HTTP/1.1 header. The value provided here must be a literal.

The VALUE attribute identifies the value to which the named header must be set. The value provided here must be a literal.

This element is used in coordination with the attributes provided for ADVANCEDSELECTOR. An HTTP request must match all the conditions identified by the ADVANCEDSELECTOR attributes before the WebAccelerator examines the request against the HEADER elements.

For example, if an invalidation object is defined using:

```
<OBJECT>
<ADVANCEDSELECTOR URIPREFIX="/apps/">
<HEADER NAME="USER_AGENT" VALUE="Mozilla/4.0 (compatible; MSIE 5.0;
Windows 98)" />
</ADVANCEDSELECTOR>
<ACTION />
</OBJECT>
```

then this HTTP request is a candidate for invalidation:

```
GET /apps/showValue.jsp HTTP/1.1
HOST: www.somesite.com
USER_AGENT: Mozilla/4.0 (compatible; MSIE 5.0; Windows 98)
```

OTHER TYPE PV_QUERY_EXP

```
<ADVANCEDSELECTOR URIPREFIX="prefix">
      <OTHER TYPE="PV_QUERY_EXP" NAME="name" VALUE="value" />
      ...
</ADVANCEDSELECTOR>
```

This element type is a Swan Labs extension to the ESI Cache Invalidation specification. This optional element identifies a query parameter. The NAME attribute provides the name of the identified query parameter. The value provided for the NAME attribute must be a literal.

The VALUE attribute provides a regular expression. The value provided for the identified query parameter must match the regular expression given here in order for the HTTP request to be a candidate for invalidation. Regular expression support is described in Appendix D, Regular Expressions.

This element is used in coordination with the attributes provided for ADVANCEDSELECTOR. An HTTP request must match all the conditions identified by the ADVANCEDSELECTOR attributes before the WebAccelerator examines the request against the conditions identified by this element.

For example, if an invalidation object is defined using:

```
<OBJECT>
<ADVANCEDSELECTOR URIPREFIX=”/apps/”>
      <OTHER TYPE=”PV_QUERY_EXP” NAME=”action” VALUE=”d.*y” />
</ADVANCEDSELECTOR>
<ACTION />
</OBJECT>
```

then these HTTP requests are candidates for invalidation:

```
http://www.asite.com/apps/bids.jsp?action=display&...
http://www.asite.com/apps/response.jsp?action=deny&.....
```

OTHER TYPE PV_UNAME_QUERY_EXP

```
<ADVANCEDSELECTOR URIPREFIX=”prefix”>
      <OTHER TYPE=”PV_UNAME_QUERY_EXP” NAME=”name” VALUE=”value” />
      ...
</ADVANCEDSELECTOR>
```

This element type is a Swan Labs extension to the ESI Cache Invalidation specification. This optional element identifies an unnamed query parameter. The NAME attribute provides the ordinal of the query parameter. Ordinals are counted from left to right in the query parameter portion of the request. Counting starts from 1.

For example, for the following URL:

```
http://www.somesite.com/apps/applicaton.jsp?action=display&twenty&forty
```

- action is a named query parameter. Its ordinal is 1.
- twenty is the value set for an unnamed query parameter. Its ordinal is 2.
- forty is the value set for an unnamed query parameter. Its ordinal is 3.

The VALUE attribute provides a regular expression. The value provided for the identified unnamed query parameter must match the regular expression given here in order for the HTTP request to be a candidate for invalidation. Regular expression support is described in Appendix D, Regular Expressions.

This element is used in coordination with the attributes provided for ADVANCEDSELECTOR. An HTTP request must match all the conditions identified by the

ADVANCEDSELECTOR attributes before the WebAccelerator examines the request against the conditions identified by this element.

For example, if an invalidation object is defined using:

```
<OBJECT>
<ADVANCEDSELECTOR URIPREFIX="/apps/">
      <OTHER TYPE="PV_UNAMED_QUERY_EXP" NAME="2" VALUE="d.*y" />
</ADVANCEDSELECTOR>
<ACTION />
</OBJECT>
```

then this HTTP request is a candidate for invalidation:

```
http://www.somesite.com/apps/bids.jsp?action=show&deny&...
```

OTHER TYPE PV_COOKIE_EXP

```
<ADVANCEDSELECTOR URIPREFIX="prefix">
      <OTHER TYPE="PV_COOKIE_EXP" NAME="name" VALUE="value" />
      ...
</ADVANCEDSELECTOR>
```

This element type is a Swan Labs extension to the ESI Cache Invalidation specification. This optional element identifies a cookie. The NAME attribute provides the cookie's name. The value provided for the NAME attribute must be a literal.

The VALUE attribute provides a regular expression. The value provided for the identified cookie must match the regular expression given here in order for the HTTP request to be a candidate for invalidation. Regular expression support is described in Appendix D, Regular Expressions.

This element is used in coordination with the attributes provided for ADVANCEDSELECTOR. An HTTP request must match all the conditions identified by the ADVANCEDSELECTOR attributes before the WebAccelerator examines the request against the conditions identified by this element.

For example, if an invalidation object is defined using:

```
<OBJECT>
<ADVANCEDSELECTOR URIPREFIX="/apps/">
      <OTHER TYPE="PV_COOKIE_EXP" NAME="PLAN" VALUE="P.*MEMBER" />
</ADVANCEDSELECTOR>
<ACTION />
</OBJECT>
```

then these HTTP requests are candidates for invalidation:

```
GET /apps/showValue.jsp HTTP/1.1
HOST: www.somesite.com
COOKIE: PLAN=PREMIUMMEMBER
      ...
```

```
GET /apps/showValue.jsp HTTP/1.1
HOST: www.somesite.com
COOKIE: PLAN=PLATINUMMEMBER
      ...
```

OTHER TYPE PV_USER_AGENT_EXP

```
<ADVANCEDSELECTOR URIPREFIX="prefix">
      <OTHER TYPE="PV_USER_AGENT_EXP" NAME="value" />
      ...
</ADVANCEDSELECTOR>
```

This element type is a Swan Labs extension to the ESI Cache Invalidation specification. This optional element identifies a value to which the HTTP USER_AGENT request header must be set in order for the WebAccelerator to consider the request as a candidate for invalidation. This value is provided using the `NAME` attribute. The `NAME` attribute takes a regular expression. Regular expression support is described in Appendix D, Regular Expressions.

This element is used in coordination with the attributes provided for `ADVANCEDSELECTOR`. An HTTP request must match all the conditions identified by the `ADVANCEDSELECTOR` attributes before the WebAccelerator examines the request against the conditions identified by this element.

For example, if an invalidation object is defined using:

```
<OBJECT>
<ADVANCEDSELECTOR URIPREFIX="/apps/">
<OTHER TYPE="PV_USER_AGENT_EXP" NAME=".*MSIE 5.0.*" />
</ADVANCEDSELECTOR>
<ACTION />
</OBJECT>
```

then this HTTP request is a candidate for invalidation:

```
GET /apps/showValue.jsp HTTP/1.1
HOST: www.somesite.com
USER_AGENT: Mozilla/4.0 (compatible; MSIE 5.0; Windows 98)
```

OTHER TYPE PV_REFERER_EXP

```
<ADVANCEDSELECTOR URIPREFIX="prefix">
      <OTHER TYPE="PV_REFERER_EXP" NAME="value" />
      ...
</ADVANCEDSELECTOR>
```

This element type is a Swan Labs extension to the ESI Cache Invalidation specification. This optional element identifies a value to which the HTTP REFERER request header must be set in order for the WebAccelerator to consider the request as a candidate for invalidation. This value is provided using the `NAME` attribute. The `NAME` attribute takes a regular expression. Regular expression support is described in Appendix D, Regular Expressions.

This element is used in coordination with the attributes provided for ADVANCEDSELECTOR. An HTTP request must match all the conditions identified by the ADVANCEDSELECTOR attributes before the WebAccelerator examines the request against the conditions identified by this element.

For example, if an invalidation object is defined using:

```
<OBJECT>
<ADVANCEDSELECTOR URIPREFIX="/apps/">
<OTHER TYPE="PV_REFERER_EXP" NAME="http://www.azippysite.com.*" />
</ADVANCEDSELECTOR>
<ACTION />
</OBJECT>
```

then this HTTP request is a candidate for invalidation:

```
GET /apps/showValue.jsp HTTP/1.1
HOST: www.somesite.com
REFERER: http://www.azippysite.com/topoftheweb.html
```

OTHER TYPE PV_HEADER_EXP

```
<ADVANCEDSELECTOR URIPREFIX="prefix">
      <OTHER TYPE="PV_HEADER_EXP" NAME="name" VALUE="value" />
      ...
</ADVANCEDSELECTOR>
```

This element type is a Swan Labs extension to the ESI Cache Invalidation specification. This optional element identifies an HTTP/1.1 header and its value. The HTTP request must have all of the headers identified here, and those headers must be set to the value identified for them, in order for the HTTP request to be a candidate for invalidation.

The NAME attribute identifies the name of a HTTP/1.1 header. The value provided here must be a literal.

The VALUE attribute identifies the value to which the named header must be set. The value provided here is a regular expression. Regular expression support is described in Appendix D, Regular Expressions.

This element is used in coordination with the attributes provided for ADVANCEDSELECTOR. An HTTP request must match all the conditions identified by the ADVANCEDSELECTOR attributes before the WebAccelerator examines the request against the conditions identified by this element.

For example, if an invalidation object is defined using:

```
<OBJECT>
<ADVANCEDSELECTOR URIPREFIX="/apps/">
```

```
     <OTHER TYPE=”PV_HEADER_EXP” NAME=”Accept-Encoding” VALUE=”.*gzip.*”
/>
</ADVANCEDSELECTOR>
<ACTION />
</OBJECT>
```

then this HTTP request is a candidate for invalidation:

```
GET /apps/showValue.jsp HTTP/1.1
HOST: www.somesite.com
Accept-Encoding: compress;q=0.5, gzip;q=1.0
```

OTHER TYPE PV_CUSTOM_HEADER_EXP

```
<ADVANCEDSELECTOR URIPREFIX=”prefix”>
      <OTHER TYPE=”PV_CUSTOM_HEADER_EXP” NAME=”name” VALUE=”value” />
      ...
</ADVANCEDSELECTOR>
```

This element type is a Swan Labs extension to the ESI Cache Invalidation specification. This optional element identifies a custom header and its value. A custom header is the unique pairing of a structured header and one of its parameters. For example, this is a structured header that might occur in an HTTP request:

```
Standards: type=SOAP;SOAP-ENV:mustUnderstand="1",version=1.2
```

A custom header is the name of the structured header combined with the name of the parameter whose value you want matched for invalidation, such as:

```
Standards:version
Standards:type
```

Each is a unique custom header, although they are based on the same structured header. The value set for a custom header is the value of its parameter as found in the HTTP request, in this case `1.2` and `SOAP`, respectively. For more information on structured headers, see “Header” on page 52.

The HTTP request must have all of the custom headers identified in `PV_CUSTOM_HEADER_EXP`, and those headers must be set to the value identified for them, in order for the HTTP request to be a candidate for invalidation. To identify a custom header in `PV_CUSTOM_HEADER_EXP`, you must already have defined a policy based on the same custom header, such as a matching policy or variation policy.

The `NAME` attribute identifies the name of the custom header in this format:

`headername``:`*`parmname`*

where *`headername`* is the name of the structured header, and *`parmname`* is the name of the parameter within the structured header that you want to affect invalidation. The `VALUE` attribute identifies the value to which the header parameter must be set. The value provided here is a regular expression. Regular expression support is described in Appendix D, Regular Expressions.

This element is used together with the attributes provided for ADVANCEDSELECTOR. An HTTP request must match all the conditions identified by the ADVANCEDSELECTOR attributes before the WebAccelerator examines the request against the conditions identified by this element.

For example, if an invalidation object is defined using:

```
<OBJECT>
<ADVANCEDSELECTOR URIPREFIX="/apps/">
     <OTHER TYPE="PV_CUSTOM_HEADER_EXP" NAME="CSP-Global-Gadget-Pref:AT"
VALUE="0" />
</ADVANCEDSELECTOR>
<ACTION />
</OBJECT>
```

then this HTTP request is a candidate for invalidation:

```
GET /apps/showValue.jsp HTTP/1.1
HOST: www.somesite.com
CSP-Global-Gadget-Pref: AT=0
```

OTHER TYPE PV_HOST_ALT

```
<ADVANCEDSELECTOR URIPREFIX="prefix">
      <OTHER TYPE="PV_HOST_ALT" NAME="value" />
      ...
</ADVANCEDSELECTOR>
```

This element type is an extension to the ESI Cache Invalidation specification. This optional element identifies a literal value to which the HTTP HOST request header must be set for the WebAccelerator to consider the request as a candidate for invalidation. Use one or more instances of this element and the ADVANCEDSELECTOR HOST attribute to define a list of hosts that a request must match. This list represents a boolean or – the value presented on the HOST request header must match at least one of the hosts identified by the ADVANCEDSELECTOR HOST attribute and this element.

The HOST value is identified using the NAME attribute. The NAME attribute takes a literal value.

For example, if an invalidation object is defined using:

```
<OBJECT>
<ADVANCEDSELECTOR URIPREFIX="/apps/" HOST="www.somesite.com">
</ADVANCEDSELECTOR>
<OTHER TYPE="PV_HOST_ALT" NAME="www.anothersite.com" />
<OTHER TYPE="PV_HOST_ALT" NAME="www.somethingelse.com" />
<ACTION />
</OBJECT>
```

then this HTTP request is a candidate for invalidation:

```
GET /apps/bid.jsp HTTP/1.1
HOST: www.somethingelse.com
```

OTHER TYPE PV_SEGMENT_L_R_EXP

```
<ADVANCEDSELECTOR URIPREFIX="prefix">
      <OTHER TYPE="PV_SEGMENT_L_R_EXP" NAME="name" VALUE="value" />
      ...
</ADVANCEDSELECTOR>
```

This element type is a Swan Labs extension to the ESI Cache Invalidation specification. This optional element identifies a path segment key or a path segment parameter. The NAME attribute identifies the path segment key or segment parameter. The value provided for the NAME attribute must be one of:

- the ordinal of the segment in the in the path as counted from left-to-right. Counting starts at 1. For example, in this path:
  `/apps/srch/magic.jsp`

  apps is path segment 1, srch is path segment 2, and magic.jsp is path segment 3.
- the ordinals of the path segment and segment parameter in the form:
  `segment_ordinal#parameter_ordinal`

  The parameter ordinal is counted from left-to-right in the segment and counting starts at 0. The segment key is ordinal 0. For example, in this path:
  `/apps/default/srch;3gYn;session2/magic.jsp`
  - srch is the segment key for segment 3. It is identified with either NAME="3" or NAME="3#0"
  - 3gYn is the value for segment parameter 1. It is identified with NAME="3#1"
  - session2 is the value for segment parameter 2. It is identified with NAME="3#2"

The VALUE attribute provides a regular expression. The value provided for the identified segment key or segment parameter must match the regular expression given here in order for the HTTP request to be a candidate for invalidation. Regular expression support is described in Appendix D, Regular Expressions.

This element is used in coordination with the attributes provided for ADVANCEDSELECTOR. An HTTP request must match all the conditions identified by the ADVANCEDSELECTOR attributes before the WebAccelerator examines the request against the conditions identified by this element.

For example, if an invalidation object is defined using:

```
<OBJECT>
<ADVANCEDSELECTOR URIPREFIX="/apps/">
      <OTHER TYPE="PV_SEGMENT_L_R_EXP" NAME="3#1" VALUE=".*A0$" />
</ADVANCEDSELECTOR>
<ACTION />
</OBJECT>
```

then this HTTP request is a candidate for invalidation:

```
http://www.asite.com/apps/srch/response.jsp;bY882A0?action=display&...
```

OTHER TYPE PV_SEGMENT_R_L_EXP

```
<ADVANCEDSELECTOR URIPREFIX="prefix">
      <OTHER TYPE="PV_SEGMENT_R_L_EXP" NAME="name" VALUE="value" />
      ...
</ADVANCEDSELECTOR>
```

This element type is a Swan Labs extension to the ESI Cache Invalidation specification. This optional element identifies a path segment key or a path segment parameter. The NAME attribute identifies the path segment key or segment parameter. The value provided for the NAME attribute must be one of:

- the ordinal of the segment in the in the path as counted from right-to-left. Counting starts at 1. For example, in this path:
  `/apps/srch/magic.jsp`

  apps is path segment 3, srch is path segment 2, and magic.jsp is path segment 1.
- the ordinal of the path segment and the ordinal of the segment parameter in the form:
  `segment_ordinal#parameter_ordinal`

  The parameter ordinal is counted from left-to-right in the segment and counting starts at 0. The segment key is ordinal 0. For example, in this path:
  `/apps/default/srch;3gYn;session2/magic.jsp`
  - srch is the segment key for segment 2. It is identified with either NAME="2" or NAME="2#0"
  - 3gYn is the value for segment parameter 1. It is identified with NAME="2#1"
  - session2 is the value for segment parameter 2. It is identified with NAME="2#2"

The VALUE attribute provides a regular expression. The value provided for the identified segment key or segment parameter must match the regular expression given here in order for the HTTP request to be a candidate for invalidation. Regular expression support is described in Appendix D, Regular Expressions.

This element is used in coordination with the attributes provided for ADVANCEDSELECTOR. An HTTP request must match all the conditions identified by the ADVANCEDSELECTOR attributes before the WebAccelerator examines the request against the conditions identified by this element.

For example, if an invalidation object is defined using:

```
<OBJECT>
<ADVANCEDSELECTOR URIPREFIX="/apps/">
      <OTHER TYPE="PV_SEGMENT_R_L_EXP" NAME="1#1" VALUE=".*A0$" />
</ADVANCEDSELECTOR>
<ACTION />
</OBJECT>
```

then this HTTP request is a candidate for invalidation:

```
http://www.somesite.com/apps/srch/response.jsp;bY882A0?action=
display&...
```

OTHER TYPE PV_EXT_EXP

```
<ADVANCEDSELECTOR URIPREFIX="prefix">
      <OTHER TYPE="PV_EXT_EXP" NAME="value" />
      ...
</ADVANCEDSELECTOR>
```

This element type is an extension to the ESI Cache Invalidation specification. This optional element identifies an extension. The extension is the value found in the left-most path segment after the left-most period (.). The extension identified here must be the extension used by an HTTP request in order for the WebAccelerator to consider the request as a candidate for invalidation. This value is provided using the NAME attribute. The NAME attribute takes a regular expression. Regular expression support is described in Appendix D, Regular Expressions.

This element is used in coordination with the attributes provided for ADVANCEDSELECTOR. An HTTP request must match all the conditions identified by the ADVANCEDSELECTOR attributes before the WebAccelerator examines the request against the conditions identified by this element.

For example, if an invalidation object is defined using:

```
<OBJECT>
<ADVANCEDSELECTOR URIPREFIX="/apps/">
<OTHER TYPE="PV_EXT_EXP" NAME="GIF|gif" />
</ADVANCEDSELECTOR>
<ACTION />
</OBJECT>
```

then this HTTP request is a candidate for invalidation:

```
http://www.somesite.com/images/up.gif
```

Chapter 13

Connection Mapping

Managing Connections
Connection Types
Connection Mapping and the Request Type Hierarchy
Connection Example

The WebAccelerator manages two different types of connections:

- connections from a user's browser to the WebAccelerator used for requesting content
- connections from the WebAccelerator to your origin servers used for proxying for content

Most of the time, these connections are independent. The WebAccelerator creates or reuses connections to your origin servers as needed to service client requests as efficiently as possible.

Because sites often use a pool of web or application servers, this default behavior may not be appropriate. This is true if your site is architected such that all the requests received for a given session must be handled by the same physical web or application server. In these cases, you need to specify an appropriate type of connection mapping for the WebAccelerator to use.

The connection settings you can specify only apply to connections used for requests. When you specify connection settings for a node, they are only used in handling requests that match the node. They are irrelevant to any responses that match the node.

Managing Connections

You manage connection policies using the Policy Editor:

1. Log into the Admin Tool.
2. Go to the Policies screen.
3. Click the Edit link for the policy set whose connection policies you want to change.
4. Select the Connections link.

For details on the screen you use to define connection policies, see the online help.

Connection Types

You identify one of three types of connections:

- Default Connection Type
- One-to-One Connection Type
- Request-based Connection Type

Regardless of the type of connection you choose, the WebAccelerator only creates new connections to your origin servers when it needs them. The WebAccelerator needs connections to your origin servers only when it proxies for content.

Default Connection Type

The default connection type is one where there is no mapping between the client connections and the WebAccelerator origin server connections.

If you use the default connection type, the WebAccelerator creates a single pool of connections to your origin servers. As new client connections arrive and the WebAccelerator must proxy for fresh content to service those client connections, the WebAccelerator reuses connections from this pool.

One-to-One Connection Type

If you specify the one-to-one connection type, there is a unique connection to your origin servers for each unique client connection. When the WebAccelerator proxies for content to service a request, it creates a unique connection used only to service requests coming from that client connection. Because most browsers create two connections when browsing a site, it is likely that the WebAccelerator creates two or more unique connections to your origin servers for each client.

When the client drops its connection to the WebAccelerator, the WebAccelerator drops the corresponding connection to its origin servers. If the client subsequently reconnects to the WebAccelerator, the WebAccelerator creates, as needed, new unique connections to your origin servers.

The one-to-one connection type is the most secure, but also the slowest. One-to-one connection types are required for authenticated (NTLM) connections. All NTLM-authenticated connections use one-to-one connections by default.

Request-based Connection Type

Request-based connections allow you to pool and reuse origin server connections to service multiple client connections. Connections in a pool are mapped to a request based on elements found in that request. You specify which elements in a request should be used to map to a connection, ensuring that all requests with certain parameters are always mapped to a particular server.

This type of connection is most useful for sites that want all session traffic sent to the same physical origin server in a pool of servers. For example, suppose your application server sets a cookie that carries information about the origin server, such as server ID. If all future requests with that same cookie value must be sent to the same application server, you can create a connection mapping rule based on that cookie to ensure the WebAccelerator proxies requests to the correct application server.

Note: Request-based connections are optimized for situations, such as server IDs, with relatively few distinct request values determining which connection to use. For other types of requests, such as those determining connections based on session IDs, you get better performance using one-to-one connections instead of request-based connections.

Managing Request-based Connections

Request-based connection policies are based on HTTP Request Data Type Parameters. To create a rule, you specify the parameter to look for in a request and the value it must have for a match to the rule. You also specify the WebAccelerator connection behavior if the request matches. The HTTP data type parameters on which you can base request-based connection rules are:

- Protocol
- Host
- Path
- Extension
- Method

- Query Parameter
- Unnamed Query Parameter
- Path Segment
- Cookie
- User Agent
- Header
- Client IP

For Path, you can specify either a full path or a path prefix. See "Paths As Prefixes" on page 68.

Value Groups

Instead of specifying a single value rule for a connections parameter, you specify a value group. A value group is a collection of value rules. The main purpose of a value group is to enable you to specify several rules for the same parameter, each with its own set of values indicating a certain behavior.

For example, a matching policy based on a cookie can only specify a single set of values for the cookie and whether that is a match or not a match. A request-based connection policy can specify a value group for a cookie, such as `sessionID`. For example, suppose your site has three application servers and your session IDs are structured such that each session ID begins with a specific application server numerical ID followed by a period followed by a session number:

```
1.AA0B3Ef72l
2.H86jT2i284K
3.b31u40FNLsms
```

You can create a value group for the `sessionID` cookie consisting of several rules:

- when cookie `sessionID` begins with `1.`, use a unique connection pool
- when it begins with `2.`, use a unique connection pool
- when it begins with `3.`, use a unique connection pool
- for all other values, use the same connection pool

As you can see, you can specify different behaviors for different values, instead of a single behavior. This is the flexibility a value group allows.

The above value group means that for any requests where `sessionID` begins with `1.`, which indicates the request should be directed to application server 1, the WebAccelerator should use a unique connection pool. Only requests directed to this server with the `sessionID` cookie use this pool. Unique pools are also used for requests where `sessionID` is `2.`, and `3.`. All other values, where `sessionID` does not represent a valid application server, can share the same connection pool.

Connection Mapping and the Request Type Hierarchy

Connection policies are organized using the Request Type Hierarchy. The Request Type Hierarchy is described in Chapter 5, The Request Type Hierarchy.

Each leaf node in the Request Type Hierarchy has one connection policy defined for it. This policy identifies the connection type that is used for requests that map to that node in the hierarchy. The connection type only applies when the WebAccelerator proxies to your origin server for fresh content.

Connection policies inherited from a parent node are inherited in their entirety. Unlike other types of caching policies where you can add to an inherited policy, you can only override an entire inherited connection policy.

Multiple Rules-based Parameters

If a request element matches parameters in two or more rules, the WebAccelerator takes the union of the matching parameter values to determine which connection pool to use. If the rules that match specify different pool behavior (one indicates using the same pool and the other indicates using a unique pool), a unique connection pool is used.

Connection Example

Suppose your site receives the types of requests described in "Request Type Hierarchy Example" on page 60 and your Request Type Hierarchy looks like:

```
-Applications
  -Default
  -Search
-Home
-Images
```

Suppose that your site uses session IDs to ensure that all requests in a given session are sent to the same physical origin server, but this is only important for your Applications and your Home page. All requests for images from your site are sent to a specific origin server identified by a unique subdomain, regardless of the session ID information. Also, your site carries session ID information in the form of a cookie called `sessionID`.

Tasks

Add a connection policy to your `Applications` node and another to your `Home` node. These policies are identical. In them, indicate that the connection type is One-to-One.

You do not have to set a connection policy for the `Images` node because the default connection type is appropriate for this node. In this example, all requests for image data are sent to the image server because of the domain information that appears on those requests.

Actions

The policies that you create for the `Applications` and `Home` nodes are identical, so this example only shows the creation of the policy on the Application node. To create the connection policy:

1. Select the Applications node in the Request Type Hierarchy.
2. Select the Connections link. Click on the `One-to-one connection mapping` button.
3. Click the Save button.

You must publish your policies in order for the WebAccelerator to use this connection policy. For more information on publishing policies, see "Publishing Policies" on page 45.

Chapter 14

Setting Responses Cached Policies

Managing Responses Cached Policies ◂
Content Assurance ◂
Authorized Content ◂
Response Codes ◂
Responses Cached and the Request Type Hierarchy ◂
Responses Cached Example ◂

Most types of policies are based on information seen in a request, such as whether a particular query parameter is set to a certain value, or whether a cookie is present or absent. Managing responses policies are based instead on information seen in the response. These rules tell the WebAccelerator whether to cache or not based on these HTTP response characteristics:

- response status codes
- response content
- whether the response contains authorized content

You create rules based these characteristics that prevent responses which normally are cache-able based on other policies from being cached by the WebAccelerator. When a response matches to the node for which these rules are defined, the response is evaluated based on the rules and cached or not cached.

Managing Responses Cached Policies

You manage responses cached policies using the Policy Editor:

1. Log into the Admin Tool.
2. Go to the Policies screen.
3. Click the Edit link for the policy set whose response caching policies you want to change.
4. Select the Responses Cached link.

For details on the screen you use to define the policies, see the online help.

Content Assurance

The WebAccelerator caches content only if it considers the content to be complete. What defines complete differs depending on the type of data contained within the response. For HTML pages, by default the response must contain both the beginning and ending HTML tags. You can override this behavior by deselecting the Content Assurance option under Responses Cached.

If you override the default behavior, the WebAccelerator determines if a response is complete based on information it finds in the HTTP response headers. For more information, see "Caching HTTP Responses" on page 193.

Authorized Content

The WebAccelerator is capable of managing requests for NTLM-guarded content. NTLM is the NT LanMan protocol, and it is widely used by the Microsoft NT family of products to safeguard sensitive information.

By default, the WebAccelerator does not cache content if it was requested over an NTLM connection. However, if you want NTLM-authenticated content to be stored in the WebAccelerator's cache, check the Authorized Content option under Responses Cached.

Response Codes

By default, the WebAccelerator only caches content that it receives from your origin servers with an HTTP response code of 200 (OK), 300 (Multiple Choices), and 301 (Move Permanently). Not all responses with 200 series codes are cached, only those with codes 200, 201, 203, and 207.

In some cases, your site might receive valid requests that result in a response code other that 200. This is most likely to occur for 300-series response codes which are used to identify an HTTP redirect.

You can use responses cached policies to cause the WebAccelerator to cache or not cache responses received with these codes:

Code	Definition
300	Multiple Choices
301	Moved. The requested content has been permanently assigned a new URI. The server is responding with a redirect to the new location for the content.
302	Found. The requested content temporarily resides under a different URI. The redirect to the new location MIGHT change.
307	Temporary Redirect. The requested content temporarily resides under a different URI. The redirect to the new location CAN change.
410	Gone. The requested content is no longer available and a redirect is not available.

Responses Cached and the Request Type Hierarchy

Responses cached policies are organized using the Request Type Hierarchy. The Request Type Hierarchy is described in Chapter 5, The Request Type Hierarchy.

Each leaf node in the Request Type Hierarchy has none or one responses cached policy defined for it. This policy identifies which responses the WebAccelerator is allowed to cache, based on response code, content completeness, and NTLM conditions.

Responses cached policies are inherited from a parent node in their entirety. Unlike other types of caching policies where you can add to an inherited policy, responses cached policies only allow you to override the entire policy defined at a parent node.

Responses Cached Example

Suppose your site receives the types of requests described in "Request Type Hierarchy Example" on page 60 and your Request Type Hierarchy looks like:

```
-Applications
  -Default
  -Search
-Home
-Images
```

Suppose that in the past you had an application requested as:

```
http://www.somesite.com/apps/magic.jsp
```

but due to current development on your site, it is temporarily requested as:

```
http://www.somesite.com/apps/magicact.jsp
```

As a result, requests for the first URL are handled with a temporary redirect to the second location.

Tasks

Add a responses cached policy to your Default node so that it is inherited by all leaf nodes. In it, indicate that response code 307 should be cached.

Actions

To create the responses cached policy:

1. Select the Default node in the Request Type Hierarchy.
2. Click 307 in the Response Codes Cached selection area.
3. Click the Save button.

You are now done creating this policy. You must publish your policies in order for this change to be used by the WebAccelerator. See "Publishing Policies" on page 45 for more information.

Chapter 15

Setting Log Formats

Defining Hit Log Formats
Pre-defined Hit Log Formats
Log Headers
Custom Log Formats
Hit Log Format Example

WebAccelerators create a log file called hit logs (because they log cache hits and misses). These hit logs are meant to contain the same sort of information that your web servers create in their log files.

Many sites perform traffic analysis against the log files generated by their web servers. To make this analysis easier to perform, many sites also have custom scripts that manipulate these files in various ways.

You can tailor the information that appears in the WebAccelerator hit logs so they work seamlessly with whatever analysis tools you use on the logs written by your web servers. Currently, you can only do this for custom policy sets. The pre-defined policy sets include pre-defined log formats and cannot be changed.

The WebAccelerator provides a Log Mover that automatically collects your hit logs from your Accelerators, transfers them to the Management Console, merges and sorts them, and places them in a location you specify for analysis and archival by your site. See the *Administration Guide* for more information.

Defining Hit Log Formats

You manage your hit logs by using the Policy Editor:

1. Log into the Admin Tool.
2. Go to the Policies screen.
3. Click on the Edit link for the policy set for which you want to configure logging.
4. In the Policy Editor, select the Log Format link. For details on using this screen, click the Help button.

You use this screen to identify:

- whether you want the WebAccelerator to log both HTTP and HTTPS requests together or log them separately

  If you log them separately, you can customize the WebAccelerator's logging behavior for each protocol individually. For example, you could describe a specific log file format for HTTP requests while not logging HTTPS requests at all.
- for each supported protocol, whether you want the WebAccelerator WebAccelerators to either:
  - log nothing,
  - log all requests using that protocol, or
  - log only those requests that the WebAccelerator was able to service from its cache
- the format of the line written to the hit log
- information you want written as the first line in the hit log file (the header line)

Pre-defined Hit Log Formats

You identify the format of your hit log files using the `Log Format` drop down list. This list contains a selection of pre-defined log formats. It also identifies any custom log formats that you might have created.

The pre-defined formats from which you can select are:

- NCSA Common (No header)
- NCSA Common (Netscape Header Format)
- NCSA Combined (No Header)
- NCSA Combined (Netscape Header Format)
- W3C Extended

Common, Combined, and Extended Formats

The Common, Combined, and Extended log formats represent a collection of logging information that is very commonly used by web servers. NCSA Common causes this information to be logged using the format:

```
host rfc931 username [date:time UTC_offset] "method URI?query_parameters
protocol" status bytes
```

For example:

```
125.125.125.2 - - [10/Oct/2002:23:44:03 -0600]
"GET /apps/example.jsp?sessionID=34h76 HTTP/1.1" 200 3045
```

NCSA Combined causes this information to be logged:

```
host rfc931 username [date:time UTC_offset] "method URI?query_parameters
protocol" status bytes "referrer" "user_agent" "cookie"
```

For example:

```
125.125.125.2 - - [23/Oct/2002:23:44:03 -0600]
"GET /apps/example.jsp?sessionID=34h76 HTTP/1.1" 200 3045
"http://www.swanlabs.com" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.0)" "UserID=ssarette;Category=PDA;Selection=Various"
```

W3C Extended causes this information to be logged using the format:

```
date time rfc931 username host method URI query_parameters status bytes
request_length time_taken protocol user_agent cookie referrer
```

For example:

```
2002-10-23 23:44:03 205.47.62.112 - 125.125.125.2 GET /apps/example.jsp
sessionID=34h76 200 3045 124 138 HTTP/1.1
Mozilla/4.0+(compatible;+MSIE+6.0;+Windows+NT+5.0 UserID=
ssarette;Category=PDA;Selection=Various http://www.swanlabs.com
```

Log Headers

Log headers are lines that appear at the top of a log file. They are used to identify the type and order of the information written to each line in the log file. These header lines are important if your log analysis software uses them to determine how to parse the log file.

There are three different schemes typically used for the formatting of log headers:

- no header line

  This is what Apache web servers do. Note that Apache servers by default write access logs that are identical to the NCSA Common format.

- NCSA Common or Combined headers

Netscape servers, and their descendants (such as the iPlanet Enterprise Server), write a header line that is unique to these family of servers. These servers generally write either the NCSA Common or Combined log format. The header lines are comprised of keywords that look like this:

```
#format=%Ses->client.ip% - %Req->vars.auth-user% [%SYSDATE%] ....
```

- W3C headers

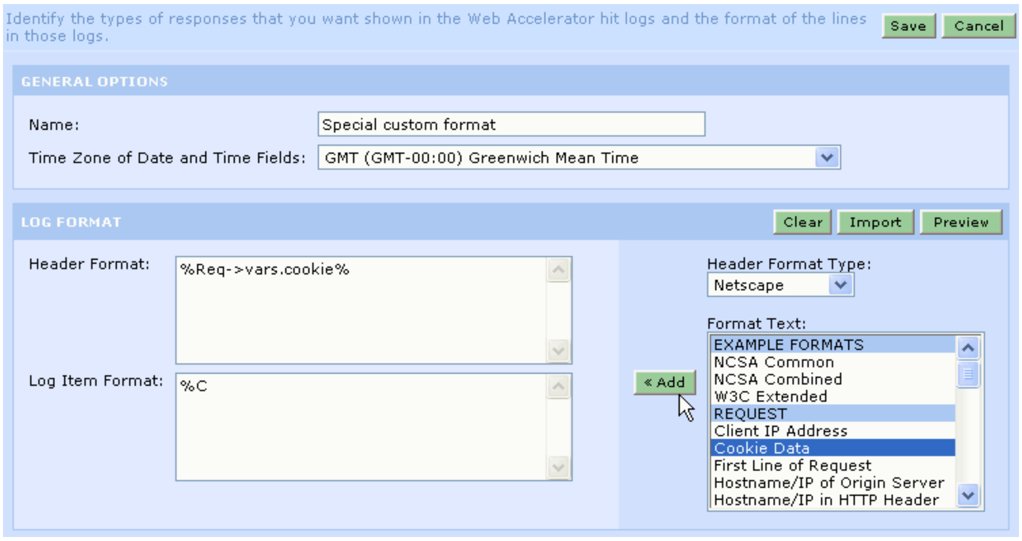
  These are defined by a W3C working draft on the extended log file format. Most Microsoft IIS web servers write the extended log file format and write W3C headers.

  The W3C working draft on the extended log file format can be found at: http://www.w3.org/TR/WD-logfile.html

If the header line that your web server writes does not conform to any of the conventions described here, and if that header line is important to you, you can cause the WebAccelerator's WebAccelerators to write a custom log header by defining a custom log format.

Custom Log Formats

WebAccelerator WebAccelerators are capable of logging a great deal more information than is available in the traditional, pre-defined log formats described in the previous sections. You can define a custom log line format, or define a custom log header line, by clicking the Edit or New button that is next to the HTTP Log Format selection list.


Identify the types of responses that you want shown in the Web Accelerator hit logs and the format of the lines in those logs.

Name: Special custom format

Time Zone of Date and Time Fields: GMT (GMT-00:00) Greenwich Mean Time

Header Format: %Req->vars.cookie%

Log Item Format: %C

Header Format Type: Netscape

Format Text:

Once you have accessed the Edit Log Format screen, you can:

- define a custom header format using the Header Format Screen

  The WebAccelerators write the exact string that you place in this box to each hit log that they write.
- define a custom log line

  You do this by specifying WebAccelerator logging codes in the Log Item Format box. A selection box is provided to the left of the screen to help you obtain the codes. Click the Help button at the top of the screen to see a complete reference on the WebAccelerator logging codes.
- import a header format

  Importing a header format allows you to paste a header format from one of your pre-existing log files into the box on the screen. Provided that the header line you use conforms to either the Netscape or W3C conventions, the Admin Tool auto-populates the Log Item Format box with the appropriate WebAccelerator logging codes.

For information on defining custom log formats, see the online help.

Hit Log Format Example

Suppose you want your hit logs to contain:

- IP of the requesting client
- date and time of the request
- HTTP method used for the request
- query URI
- query parameters passed using the GET method
- HTTP version used by the request
- HTTP status returned on the response
- length of the response
- actual origin server used for any proxies that the WebAccelerator might have made as a result of the request

Your origin servers are all running Apache, so you want your log lines to follow the NCSA Common (No Header) formatting convention:

```
xxx.xxx.xxx.xxx [date:time UTC_offset] "Method uri?query_parameter
HTTP_ver" response_status response_length
```

However, you want one additional field, the origin server identification information:

1. Log into the Admin Tool.
2. Go to Policies and click the Edit link for the policy set for which you want to configure logging. On the Policy Editor screen, select the Log Format link.
3. In the Logs section, from the Log Format drop-down list, select `NCSA Common (no header)`.
4. Click the Edit button to edit this format.
5. In the Edit Log Format screen, provide a meaningful name and select your preferred time zone. The time zone you pick is used to calculate the time and date information written to the log, including the UTC offset value.
6. In the Format Text drop-down list, select `Hostname/IP of Origin Server` and click the Add button.
7. The additional field (`%B`) appears in the Log Item Format box. You might have to manually add a space before the formatting code.
8. Click the Save button.
9. You now see the new custom log format selected for the Log Format drop-down list.
10. Click the Save button.

You are done creating your custom log format. You must publish your policies in order for this change to be used by the WebAccelerator. For more information on publishing policies, see "Publishing Policies" on page 45.

Appendix A

Requirements for Cache Behavior

Servicing HTTP Client Requests
Servicing HTTP Client Requests from Cache
Caching HTTP Responses

The WebAccelerator follows certain well-defined rules when:

- attempting to service an HTTP client request
- servicing an HTTP client request from its cache
- caching an HTTP response that it receives from your origin servers

This appendix describes these rules in detail.

Servicing HTTP Client Requests

In order to maintain high performance, requests must meet certain conditions before the WebAccelerator attempts to service them from the cache. Client requests must conform to these constraints:

- requests must provide a request header that identifies the method, URI, and protocol. This information must appear in the first line of the request header.
- requests must provide the targeted host using the HOST request header. The identified host must be mapped to an origin server at your site.
- method must be a GET, POST, or HEAD
- protocol must be HTTP/0.9, HTTP/1.0, or HTTP/1.1
- the HTTP request header can be no larger than 8192 bytes
- the HTTP post data on the request can be no larger than 32768 bytes
- if the request provides the Expect request header, the only allowable value is `100-continue`
- if the request provides the Content-Type request header, the value must be `application/x-www-form-urlencoded`

If the HOST header is missing, or its value is unknown to the WebAccelerator, the WebAccelerator responds to the requesting client with a 400-series error message. If the request violates any of the other conditions, the WebAccelerator redirects the request to your origin servers for processing.

Servicing HTTP Client Requests from Cache

When an WebAccelerator receives an HTTP request, it must decide whether it can service the request from content stored in its cache, or whether it must proxy the request to your origin servers. The WebAccelerator makes this decision using this process:

1. An WebAccelerator receives an HTTP request that meets the conditions described in "Servicing HTTP Client Requests".
2. The WebAccelerator performs application matching against the request and retrieves the appropriate caching policies. Application matching is described in "Application Matching Policies" on page 7.
3. The WebAccelerator determines if there is a proxying policy in effect for the request. If so, the WebAccelerator proxies the request to your origin servers. Proxying policies are described in Chapter 10, Setting Proxying Policies.

4. If there is no applicable proxying policy, the WebAccelerator retrieves the appropriate compiled response from its cache. If no such compiled response is available, the WebAccelerator proxies the request to your origin servers.
5. If a compiled response is found, the WebAccelerator determines if there is a content invalidation rule raised for the compiled response. There are several mechanisms that could raise a content invalidation rule. See Chapter 12, Invalidating Cached Content for details.
6. If a content invalidation rule is raised for the compiled response, the WebAccelerator examines the rule's effective time against the compiled response's last refreshed time. If the compiled response's last refreshed time is before the rule's triggered time, the WebAccelerator proxies the request to your origin servers.
7. If there is no raised content invalidation rule, or if the compiled response's last refreshed time is after the invalidation rule's effective time, the WebAccelerator examines the compiled response's time-to-live information to see if the compiled response has expired. If it has expired, then the WebAccelerator proxies the request to your origin servers.
8. If the compiled response has not expired, the WebAccelerator services the request using the information it finds in the compiled response.

Caching HTTP Responses

In order to maintain high performance, responses from your origin servers must meet certain conditions before the WebAccelerator can cache them. Responses that the WebAccelerator does not cache are forwarded to the originating requestor.

The response must meet these requirements in order to be cached:

- no do-not-cache policy can be in effect for the corresponding request. See Chapter 10, Setting Proxying Policies for more information.
- first line of the response must identify the protocol, a response code that is an integer value, and a response text. For example:
  `HTTP/1.1 200 (OK)`
- if the Transfer-Encoding response header is used on the response, the value can only be `chunked`
- the response must be complete. The WebAccelerator decides what constitutes a complete response depending on the type of the response:
  - for an HTML page, the response must contain both the beginning and ending HTML tag

    This behavior can be disabled using the matching HTML option. You set that option in the Policy Editor. Select the Responses Cached link and then select Content Assurance. If disabled, the HTML pages are eligible to be cached.

  - for any type other than an HTML page, the response body size must match the value specified on the Content-Length response header

    If Content-Length is not used, then chunked transfer encoding must be used. If chunked transfer encoding is used, the WebAccelerator must receive the final zero-sized chunk before it considers the response to be complete. See section 3.6 in the HTTP/1.1 specification for information on chunked encoding:
    http://www.w3.org/Protocols/rfc2616/rfc2616-sec3.html#sec3.6

- the body of the response must not exceed the value set for the `maxResponseDataSize` parameter in the WebAccelerator's configuration file. By default, this value is 2Mb. However, it can be modified by the personnel responsible for managing your WebAccelerator installation.

If the response received from your origin servers does not conform to these conditions, the response is not cached but simply redirected to the original requesting client.

Appendix B

Rewrite Engine

Customizing Rewrite Scripts ◂
Running Rewrite Scripts ◂
Language Processing ◂
Language Expressions ◂
Procedures ◂
Language Literals ◂
Predefined Procedures ◂
Transform ◂
Obtaining Information from the Request ◂

The WebAccelerator Rewrite Engine provides a procedural language for manipulating HTTP responses received from your origin servers. This manipulation occurs before the response is processed by the WebAccelerator, so it is the manipulated response that the WebAccelerator manages and caches, not the actual response sent by your origin servers.

You can use the Rewrite Engine to modify your site's behavior and response without actually changing the code that generates your site. This is particularly useful for sites that might want to modify the HTTP response headers presented to an WebAccelerator. Sometimes it is easier to substitute parameter values using the Rewrite Engine than to use the parameter value substitution feature of assembly policies.

The Rewrite Engine can also be used to introduce ESI to a site's pages without modifying the code that generates those pages.

Customizing Rewrite Scripts

There is one rewrite script archive for your WebAccelerator system, stored as a zip file. All the rewrite scripts that you want to make available to your WebAccelerator system must be in this single zip archive. If you want to customize or add a rewrite script, you must download the archive, make your changes, and upload it. The uploaded file replaces the existing archive. For example, if you upload a single script, it replaces all the pre-loaded rewrite scripts that were shipped with the WebAccelerator.

The scripts you create using the Rewrite Engine must be uploaded before they are available to the WebAccelerator. To add new scripts you have created, or modify existing rewrite scripts:

1. Go to the Policies screen.
2. Click the Import/Export Transforms link. A pop-up window appears.
3. In the Export section, download the archive to your desktop.
4. On your desktop, unzip the archive to a folder. You can use either WinZip or the Linux zip utility to zip and unzip the file.
5. Copy your new rewrite scripts into the folder, or modify one or more of the existing scripts that you downloaded.
6. When you are done with your changes, use WinZip or the Linux zip utility to recreate the archive zip file. Do not include subdirectories. Include all the scripts that were in the original archive, unless you are specifically deleting a script and it is not used in any of the policy sets or object type definitions.
7. Place this new zip archive where it accessible from your browser.
8. Go back to the Policies screen.
9. Click the Import/Export Transforms link. A pop-up window appears.
10. In the Import section, browse to the zip file containing the rewrite scripts and click the Upload button. This uploads the new archive, which completely replaces the existing archive.

Once the scripts are uploaded, they are available to the WebAccelerator and can be specified in assembly policies.

Warning: Use caution when you change the rewrite script archive. If you delete a script that had been loaded and available to the system, there is no indication whether existing policies or object type definitions use the script. If the Rewrite Engine tries to run a missing script, the response is simply not rewritten.

Running Rewrite Scripts

Rewrite scripts are only run against responses that result from a proxy for fresh content. A script is never run against a response taken from cache. This is useful in several ways:

- it means responses are rewritten before they are cached, so it is the rewritten response that is cached. The rewrite script only has to execute once, when the response is first received from the origin servers.
- it also means that you can use a script to change a response so that it is more or less likely to be cached by the Accelerator. For example, you can rewrite or add response headers to change values in Content-Type or Cookie headers.

How a Rewrite Script Is Run

A rewrite script is run when a response is matched to an object type or assembly policy that specifies a rewrite script. The decision process used for running a rewrite script is:

1. A request is received by the WebAccelerator, and the WebAccelerator determines it must proxy for a response because it cannot service the request from cache.
2. The response is received by the WebAccelerator, and the response is classified by object type and the content type is generated. This process is described in "Classifying Responses" on page 213.
3. If the object type definition includes the `tranformFile` option, which specifies a rewrite script, this rewrite script is run.
4. The WebAccelerator matches the response to a node, taking into consideration its content type. If the response has a different content type than its request did, and there are matching rules based on the Content Type parameter, the response might match to a different node than the request.
5. The WebAccelerator looks at the assembly policy for the node to which it matched the response, to see what assembly options are specified.
6. If the object type definition did not include a rewrite script, the WebAccelerator looks at the rewrite assembly option. If a rewrite script is specified, it is run. If the object type definition included a rewrite script, the rewrite assembly option is ignored.
7. The WebAccelerator completes its response handling, which includes sending the response to the client and if possible, caching the response.

Specifying a Rewrite Script to Run

You can only specify one script to be run against any particular response. If there are several procedures you want to perform against a response, you must combine them into one script.

There are two ways to specify a rewrite script:

1. in the object type definition, found in the global configuration fragments file
2. in an assembly policy

The most common way to specify a rewrite script is in an assembly policy, because you create and edit assembly policies in the Admin Tool and because you can customize this setting for each node in each policy set. For a description of how to specify a rewrite script in an assembly policy, see "Rewrite Engine Scripts" on page 90.

Only a few object type definitions include a rewrite script because:

- an object type definition applies across the entire WebAccelerator, regardless of application or policy set
- a rewrite script specified in an object type overrides any rewrite setting in an assembly policy

Only specify a rewrite script in an object type definition if you want to ensure the script is used for any response matching that object type across your entire system, and you do not want an assembly policy to ever affect this setting. For information on specifying a rewrite script in an object type definition, see the *Administration Guide*.

If you want to specify a rewrite script for a particular object type but do not need to make this a global setting, create a node in your policy set with a matching policy that specifies the Content Type for the object type. Then create an assembly policy for the node that specifies the rewrite script. All responses that match the object type are matched to this node and the rewrite script specified in the assembly policy is run.

Language Processing

As the script is run against the response, the Rewrite Engine generates an emit list. This is a list of data that the Rewrite Engine sends to the WebAccelerator for processing. Most of the origin server's response is simply stored in the emit list.

You use the Rewrite Engine's programming language to add, change, or delete information from the emit list, essentially changing the response header and the page that the WebAccelerator caches and sends. After the script has completed processing, the Rewrite Engine sends the modified response to the WebAccelerator.

Language Expressions

The Rewrite Engine uses a prefix-based language. This means that the operator comes before the operands. For example:

`+ 3 5`	Adds 3 and 5.
`== a b`	Tests whether a is equal to b.
`- (+3 5) 4`	Adds 3 and 5 and subtracts 4.
`setvar a "foo"`	Sets variable a to the string constant "foo".
`print(+ 3 5);`	Prints the sum of 3 and 5.

Use parentheses to denote a unit of execution where a literal might otherwise be expected. For example, if you want to print a string, you use:

```
print "this is a string";
```

However, if you want to print a string that is returned by a function, use:

```
print ( myFunc );
```

Procedures

You can define procedures to perform whatever custom operations you require. The syntax of a procedure is:

```
Proc( (argument1 argument2 argument2 ...)
      ( statement1;
        statement2;
        ...
        statement n;
       ) );
```

If you want to call a procedure as a function, provide a handle to the procedure using `setvar`:

```
setvar myFunction Proc( (aVariable)
             (print "myFunction works";
              print aVariable; ) );
```

Language Literals

The Rewrite Engine provides these built-in tokens:

Arithmetic

```
+, -, *, /, ( , ), =
```

Comparison

```
==, !=
```

Conditionals

```
if, ifelse
```

For example:

```
if ( == a b) (print "a and b are equal");
ifelse ( == a b) (print "a, b are equal") (print "a, b not equal");
```

Functions

The Rewrite Engine provides these built-in functions:

return <value>

Returns a value generated by a procedure.

print <value>

Prints a value to the console.

setvar <var_name> <value>

Declares a variable. The variable value can be a constant or a procedure declaration. If a procedure declaration is specified, the variable becomes a handle for the procedure.

```
setvar myFunc Proc (arg1) ( print arg1; ) );
myFunc "this string gets printed";
```

place <value>

Allocates a memory location for the provided data. You can set a variable to reference this location and at a later date modify, emit, or print that data.

```
setvar myHandle (place "this handle’s data");
```

fillplace <place> <value>

Resets the place's value to contain the provided data.

```
fillplace myHandle "changed handle data"
```

viewplace <place>

Returns the contents of a place.

```
setvar myHandleContent ( viewplace myHandle );
```

import <filename>

Includes and executes the contents of another file that contains Rewrite Engine code. The imported code is executed at the point where it is imported into the file.

bindarg <func> <index> <value>

Sets a specified argument for a function to a static value and returns a function that no longer requires you to specify that argument. You identify the function argument that you want to bind by its numerical position in the argument list (counting starts at 0).

```
setvar printit Proc( (arg1 arg2)
       (print arg1; print arg2;) );
setvar showit (bindarg printit 1 "foobar");
printit "my first argument" "my second argument";
showit "showit first argument";
```

string_length <string>

Returns the length of the identified string.

```
print( string_length "Some random string.");
```

string_join <str1> <str2>

Returns a single string that is the combination of the two strings provided to the function.

```
print( string_join "Some random " "string");
```

string_substring <str> <from> <to>

Returns the substring that begins and ends at the identified indexes. The first character of a string is at index 1.

```
print( string_substring "Some random string" 5 11 );
(prints the word 'random').
```

string_indexof <str1> <str2>

Returns the index of a substring within a string. -1 is returned if the substring does not exist in the string.

```
setvar theString "Some random string".
setvar random_len (string_length "random");
setvar random_begin (string_indexof theString "random");
print( string_substring theString random_begin
     (+ random_begin random_len ) );
```

span <left> <right>

Keyword denoting a substring as returned by a `search_*` function. The span is the actual value that is found by the search function. Each span also has a left and right value. These are integers representing the left and right index within the larger string (either the response header or the response body) where the span was found.

mfile

Causes memory to be allocated for general storage purposes. Currently, this is only used with the `os_pipe` function to collect data returned to standard out by the external process. See the `os_pipe` description below for more information.

```
setvar memLocation (mfile);
```

span_left <integer>

Returns the integer that represents the left value of the span.

```
print( myspan span_left);
```

span_right <integer>

Returns the integer that represents the right value of the span.

```
print( myspan span_right);
```

span_setleft <integer>

Sets the span's left value to the identified integer.

```
span_setleft myspan 3;
```

span_setright <span> <value>

Sets the span's right value to the identified integer.

```
span_setright myspan 100;
```

span_init <span> <left> <right>

Sets both the left and right value for a span.

```
span_setright myspan 3 100;
```

emit_add <arg>

Adds the argument to the emit list. If the argument is a simple variable, the value of that variable is emitted at the beginning of the span that is currently being processed. If the argument is a procedure, then the procedure is executed at the beginning of the span that is currently being processed. This function is usually used in conjunction with `search_addtoken` or `search_addpriority`.

emit_mergespan <span>

Adds a span to the emit list. If the previous element in the emit list is an adjacent span, the two spans are merged into a single emit list entry. This greatly improves the efficiency of the program. In all other respects, this function works identically to `emit_add`.

emit_setautoemit <bool>

If `TRUE`, emission occurs. If `FALSE`, emission is suppressed. Use this function with `search_addtoken` to remove blocks of text from your HTTP response.

```
setvar suppress Proc( (key proc span)
        ( emit_setautoemit FALSE;) );
setvar enable Proc( (key proc span)
        (emit_setautoemit TRUE;) );
search_addtoken "<--" suppress;
search_addtoken "-!>" enable;
```

emit_getspanlength <span> <begin> <end>

Returns an integer representing the length of the span. This function takes two arguments that represent the beginning and ending of the span.

```
print( emit_getspanlength myspan 4 89 );
```

os_pipe <string> <process> <output location>

Pipes the information stored in `<string>` to the external `<process>`. Whatever information is written to stdout by `<process>` as the result of the operation is stored in `<output location>`. Currently `<output location>` must be a variable that points to an `mfile` location.

```
setvar memLocation (mfile);
os_pipe (span begin end)  "/opt/pivia/bin/someprogram" memLocation;
emit_setautoemit TRUE;
emit_add memLocation;
```

request_get <type char> <name key>

Returns a request header value. The value returned is identified by the appropriate mnemonic and key. See "Obtaining Information from the Request" on page 206 for more information.

search_addtoken <key> <func>

Causes the identified procedure to be executed when the identified key is found in the HTTP response. The procedure provided to this function must accept the arguments `key`, `proc`, and `span`. See `emit_setautoemit` for an example of usage.

search_addprioritytoken <priority> <key> <func>

Works exactly the same as `search_addtoken`, except that you can add a priority to the procedure. That is, this procedure with priority 0 operates identically to `search_addtoken`. Procedures are executed by order of priority, from the lowest integer to the highest. Negative numbers are allowed.

```
search_addprioritytoken 1 "foobar" Proc( (key proc span)
            ( emit_add "this replaces 'foobar'"; ) );
```

search_removetoken <key> <func>

Removes a token from the emit list. The result is that the Rewrite Engine no longer searches for the token. This is usually used with `search_addtoken` or `search_addprioritytoken`.

```
search_addtoken "foobar" Proc( (key proc span)
    ( search_removetoken key proc;
      emit_add "this replaces 'foobar'";
    ) );
```

Predefined Procedures

F5 WebAccelerator comes with predefined Rewrite Engine procedures that you can use to build your own procedures. You can find them on the Assembly screen in the Policy Editor. The prepackaged functions are contained in these files that you can import into your code:

Function	Description
`common.code`	Imports `cl.code`, `headers.code`, and `content.code`. Also makes these functions available: ■ `end_of_header_key` – Key that identifies the end of the response header. ■ `skip_until` – Function that causes emission to stop until a specific key is found.
`cl.code`	Importing this file causes the Content_Length response header to be recalculated and set correctly.
`headers.code`	Makes these header-related functions available: ■ `set_header` – Set the identified header to the identified value, but only if it already exists in the response. ■ `add_header` – Add the identified header to the response and set it to the identified value. ■ `remove_header` – Remove the header from the response if it exists.
`content.code`	Provides several content-related functions: ■ `insert_at_begin` – Adds the identified string to the beginning of the response's body. ■ `insert_at_end` – Adds the identified string to the end of the response's body ■ `remove_html_comment` – Removes the HTML comment that begins with a specified substring. This substring must be the text immediately following the HTML begin comment token (`"<!--"`), including leading whitespace.
`pdf.code`	Linearizes any file with a PDF MIME type. When you add this to your Accelerators, ensure this is added only to nodes that would be matched to PDF file requests. For example, add it to the assembly policy for a node with a matching policy based on extensions with the value `PDF` or `pdf`.

Transform

To help you create Rewrite Engine procedures, Swan Labs makes available the `transform` utility. This command line utility can be used to execute your procedures so that you can test them before adding them to your F5 WebAccelerator installation. This utility can be found in your WebAccelerator installation in this directory:
`/opt/pivia/dac/bin/transform`

This utility takes these command line arguments:

```
transform -c <code file> -i <input file> [-r <request file>]
-o <output_file> [-n <num iterations>]
```

where:

- `<code file>` is the file that contains your Rewrite Engine code
- `<input file>` is the file that contains the HTTP response that you want to transform
- `<request file>` is an optional parameter that contains the HTTP request that retrieves the `<input file>`
- `<output file>` is the file where you want the results of the transformation to go. Note that the `print` function does not place its output into this file -- only emit_* does that.
- `<num iterations>` is an optional parameter that identifies the number of times you want to perform the transformation

Backup: If you use the `transform` utility to create or customize procedures, you should be backing up these procedures after making any changes. The procedures are stored as a zip file in the `/opt/pivia/dac/transforms` directory. Make a copy of this file and put it on a different machine or on media such as a CD-ROM. Ensure that the people responsible for keeping the WebAccelerator running know where your backups are. They will need to use these backups to recreate the system on a standby machine.

Obtaining Information from the Request

The Rewrite Engine operates on an HTTP response that the WebAccelerator receives from your origin servers. This response was generated as the result of an HTTP request that the WebAccelerator proxied to your origin servers.

You can obtain information from this original HTTP request using the Rewrite Engine's `request_get` function. To use this function, you provide a single-character mnemonic that identifies the part of the request you want. For example, 'C' indicates that you want to retrieve a cookie.

You then provide a key that uniquely identifies the field in the part of the request that you want to retrieve. For example:

```
request_get "C" "sessionID"
```

returns the value set for the `sessionID` cookie.

These are the mnemonics that you can use with the `request_get` function:

Mnemonic	Description
A	Return the value of the User-Agent field in the request. The key that you provide for this mnemonic must always be "`agent`": `request_get "A" "agent"`
&	Return the value of an unnamed query parameter. The key that you provide must conform to the numbering convention as described in "Unnamed Query Parameter" on page 49. For example, to obtain the value of the second unnamed query parameter in the request: `request_get "&" "2"`
B	Return the body of the request. The key that you provide for this mnemonic must always be "`body`": `request_get "A" "body"`
C	Return a cookie. The key that you provide for this mnemonic must equal the name of a cookie that you expect to appear in the request: `request_get "C" "sessionID"`
E	Returns the extension on the request. The extension is the value in the request that appears between the right-most period (.) and the request's query parameters. For example, in the request: `http://www.somesite.com/index.html` the extension is `html`. The key that you provide for this mnemonic must always be "`ext`": `request_get "E" "ext"`
F	Returns the value of a form field. The key that you provide for this mnemonic must equal some field in a form expected in this request: `request_get "F" "User"`
H	Returns the value for a generic HTTP request header. The key that you provide for this mnemonic must be the name of the HTTP request header for the value you want returned: `request_get "H" "Accept-Encoding"`

Mnemonic	Description
D	Returns the domain to which this request was sent. The key that you provide for this mnemonic must always be "`domain`": `request_get "D" "domain"`

>	Returns the value of a path segment, as counted from the left. The key that you provide this mnemonic must identify a segment ordinal and a segment parameter ordinal. See "Segment Ordinals" on page 50 and "Segment Parameter Ordinals" on page 50 for a definition of these values.
	The key delimits these two values using a pound (#) sign. For example, to access the second path segment as counted from the left in the path:
	`request_get ">" "2#0"`
	To access the third segment parameter in the same path segment, again, counting from the left:
	`request_get ">" "2#3"`

M	Returns the method used for the request. The key that you provide for this mnemonic must always be "`method`": `request_get "M" "method"`
P	Returns the full path used in the request. The key that you provide for this mnemonic must always be "`path`": `request_get "P" "path"`
:	Returns the port used in the request. The key that you provide for this mnemonic must always be "`PORT`": `request_get ":" "PORT"`
Q	Returns a query parameter from the request. The key that you provide for this mnemonic must equal some query parameter expected in this request: `request_get "Q" "category"`
R	Returns the value of the request's Referer field. The key that you provide for this mnemonic must always be "`referer`": `request_get "R" "referer"`

Mnemonic	Description

<	Returns the value of a path segment, as counted from the right. The key that you provide this mnemonic must identify a segment ordinal and a segment parameter ordinal. See “Segment Ordinals” on page 50 and “Segment Parameter Ordinals” on page 50 for a definition of these values.
	
	The key delimits these two values using a pound (#) sign. For example, to access the first path segment as counted from the right in the path:
	
	`request_get "<" "1#0"`
	
	To access the third segment parameter in the same path segment, again, counting from the right:
	
	`request_get "<" "1#3"`

V	Returns the version of the protocol used for the request (1.0, 1.1, and so forth). The key that you provide for this mnemonic must always be "`version`": `request_get "V" "version"`
S	Returns the transfer protocol used for the request (either HTTP or HTTPS). The key that you provide for this mnemonic must always be "`transferProt`": `request_get "S" "transferProt"`
Y	Returns the client IP from which the request was sent. The key that you provide for this mnemonic must always be "`ip`": `request_get "Y" "ip"`

Appendix C
Response Headers

Response Processing ◀
ESI Surrogate-Control Headers ◀
X-PvControl Response Header ◀
X-PvInfo Response Header ◀

The information found in HTTP requests and HTTP request headers is used extensively to determine the behavior of your WebAccelerator. Information in the requests is used for application matching, and most policies are defined using values based on HTTP request data.

The information found in HTTP response headers is used less extensively, but is still important for determining WebAccelerator behavior. Response headers cannot influence as many aspects of WebAccelerator behavior because responses are received after certain types of processing have completed. For example, the presence or value of a response header cannot affect the UCI that was generated for the request and is also assigned to the response.

The WebAccelerator also inserts several response headers to describe how it handled a response. These headers are informational only. They cannot be changed and do not affect processing. However, they can provide useful information when you are evaluating or debugging your policies.

Response Processing

When a browser sends an HTTP request to the WebAccelerator, the WebAccelerator compares the host on the request to the host maps for the WebAccelerator system. When it finds a match, it knows which application the request is for and what policy set to use.

The WebAccelerator then performs application matching using that policy set. If possible, application matching assigns a node in the Request Type Hierarchy to the request. This node determines which policies are in effect for the request, and this in turn affects the application processing that happens next. The initial application processing following application matching includes:

- authentication/authorization processing
- content identity and content variation processing
- proxy rule processing
- connection identity and connection selection processing
- invalidation triggers processing
- session management processing

Content variation, proxy rules, connection selection, and invalidation trigger processing are controlled by the policies in the policy set assigned to the application the request was for. You can change those policies in the Policy Editor.

After this application processing occurs, the WebAccelerator determines if it can service the request from cache or if it must proxy to the origin servers. Regardless of whether the request is proxied or serviced from cache, the response that is generated is assigned the same node in the Request Type Hierarchy as the request.

Responses Served from Cache

If the request can be serviced from cache, the WebAccelerator locates the object in cache using the UCI created from elements in the request, based on the variation policies for the node assigned to the request.

The response is assembled, using any assembly policies specified by that node. Any content lifetime and responses cached policies in effect for the node are also applied to the response. The response is then sent to the client.

Responses Served from a Proxy

If a request must be proxied to the origin servers, the WebAccelerator performs some final processing on the request before sending it to the origin servers. Part of this final processing includes adding an ESI Surrogate-Capabilities header to identify itself to the origin servers.

The origin servers service the request, creating an HTTP response. If this response includes ESI markup, the origin servers should include an ESI Surrogate-Control header in the response.

Classifying Responses

When the response comes back from the origin servers, the WebAccelerator classifies the response by object type, based on the types defined in the global configuration fragment file, `globalfragment.xml`. The WebAccelerator looks at the response headers and classifies the response based on the first information it finds, in this order:

1. the file extension from the filename field in the Content-Disposition header
2. the file extension from the extension field in the Content-Disposition header
3. the Content-Type header in the response, unless it is an ambiguous MIME type
4. the extension of the path in the request

For example, if the extension in the filename field in the Content-Disposition header is empty, the WebAccelerator looks at the file extension in the extension field of the header. If that has an extension, the WebAccelerator attempts to match it to an object type in the `globalfragment.xml` file. If there is no match, the WebAccelerator assigns an object type of `other` and uses the settings for `other`. The WebAccelerator only looks at the information in the Content-Type header if there is no extension in the filename or extension fields of the Content-Disposition header.

If the WebAccelerator finds a match to an object type in the `globalfragment.xml` file, it classifies the response as being that object type, group, and category, and uses the settings for compression and rewrite scripts. These settings can override compression and rewrite script settings specified in the assembly policy for the node of the response.

Once the response is classified by object type, the content type for the response is generated by appending the object type to the group:
`group.objectType`

Application matching occurs for the second time (the first was matching the request to a node), using the new content type. In many cases, this content type is the same as the content type for the request, in which case the response matches to the same node as the request. Also, if the Content Type parameter is not used in any matching rules, the response matches to the same node as the request.

The response is matched to a node, and those policies are now used to handle the response.

Rewriting and Assembling Responses

Next, the response is processed by the Rewrite Engine. The Rewrite Engine runs one rewrite script, either:

1. the rewrite script specified in the definition of the response's object type if there is one, or
2. the rewrite script specified in the assembly policy for the node to which the response matched, if there is one

If there is a rewrite script specified in the object type definition, the Rewrite Engine ignores the assembly policy. However, most object type definitions do not specify a rewrite script.

After the Rewrite Engine is finished with the response, the response is compiled and the WebAccelerator determines if it is cache-able or not, including looking for a responses cached policy that applies to the node for the response. If it is cache-able, a copy of the response is cached.

The response is assembled using:

- the assembly policy specified by the node assigned to the response
- the compression setting in the `globalfragment.xml` file for the object type under which the response was classified. The compression setting can be either:

`none`	overrides the setting in the assembly policy. The response is never compressed.
`pivia`	overrides the setting in the assembly policy. The response is compressed whenever it is being sent to another WebAccelerator, but never compressed when sent to a browser client.
`policyControlled`	defaults to the setting in the assembly policy.

If the response includes an ESI Surrogate-Control header, ESI processing occurs as part of the assembly. This is also when parameter substitution occurs. Any lifetime policy in effect for the node is also applied to the response. The response is then sent to the client.

Note: For more information on classifying responses under object types and modifying the `globalfragment.xml` file to add or change types or settings, see the *Administration Guide*.

Response Header Processing

Response header processing generally occurs after the response is compiled. If there are any response headers that affect response caching, these are examined when the WebAccelerator makes its caching decisions, before the response is assembled. Response headers that affect assembly, such as the ESI Surrogate-Control header, are examined when the WebAccelerator begins the compile and assembly process.

When using response headers to influence WebAccelerator behavior, remember that they can only influence processing that occurs after the response has exited the Rewrite Engine. If a response header is to change which policies apply to a response, it can only affect policies for:

- responses cached
- assembly, not including any rewrite scripts or Do Not Rewrite setting because the response has already exited the Rewrite Engine
- proxying, but only how that policy affects cacheability of the response, because proxying to the origin servers has already happened
- lifetime

Response headers have no effect on application matching, variation, or invalidations.

ESI Surrogate-Control Headers

ESI is an open specification used to control the behavior of web surrogates like the WebAccelerator. ESI Surrogate-Control headers are used when a site uses ESI to define page assembly.

When the WebAccelerator receives an HTTP request that it cannot service from cache, it proxies that request to your origin servers. At the time of proxy, the WebAccelerator adds an ESI Surrogate-Capabilities header to the HTTP request.

The ESI header appears in the HTTP request as:

```
Surrogate-Capabilities: pivia=”ESI/1.0”
```

The header specifies `pivia` as the device token used by the WebAccelerator, and identifies the WebAccelerator as an ESI 1.0 compliant device.

When your origin servers respond to the WebAccelerator’s request, the response must contain an ESI Surrogate-Control response header if there is any ESI markup used in the response. The minimum this header must contain is the value `ESI/1.0`:

```
Surrogate-Control: content=”ESI/1.0”
```

If the WebAccelerator does not see `ESI/1.0` identified in the response header, it ignores any ESI markup that appears in the response.

The ESI specification that defines the Surrogate-Control response header is available on the web at the http://www.esi.org/architecture_spec_1-0.html site.

For more information on using ESI to define page assembly, see Chapter 9, Assembling Content with ESI.

Supported Control Directives

The Surrogate-Control response header allows origin servers to use control directives to dictate how surrogates should handle response entities. Control directives control processing and cache behavior. The WebAccelerator supports these control directives:

- Content Directive
- No-Store Directive
- Max-Age Directive

The Surrogate-Control header can contain one or more of these directives, separated by a comma (,).

Content Directive

The content directive is required if the HTTP response contains ESI markup. Your servers must include the Surrogate-Control header with the content directive:

```
  Surrogate-Control: content=”ESI/1.0”
```

or the WebAccelerator fails to process any ESI markup contained in the response. See Chapter 9, Assembling Content with ESI for more information.

The content directive is always specified as a name=value pair.

No-Store Directive

If the Surrogate-Control header contains the no-store directive, the WebAccelerator is directed to not cache the response.

The no-store directive is always specified simply as the string `no-store`. For example:

```
Surrogate-Control: content=”ESI/1.0”,no-store
```

Max-Age Directive

If the Surrogate-Control header contains the max-age directive, this specifies how long the content is to be cached. The value specified by max-age is used by the WebAccelerator as the compiled response TTL value. The units are expressed in seconds. For example, this indicates the content should be cached for 1 hour:

```
max-age=3600
```

A stand-in period can also be expressed with a + operator. For example, this indicates the content should be cached for 30 minutes with a 10 minute stand-in interval:

```
max-age=1800+600
```

Stand-in periods are defined in "Stand-In Period" on page 131.

Max-age can also be specified for an HTML fragment using the max-age parameter on an esi:inline statement. See "esi:inline" on page 108 for more information.

Unsupported Control Directives

The other control directives you can define in the Surrogate-Control response header are not supported by the WebAccelerator. If the WebAccelerator finds these directives in a Surrogate-Response header, it ignores them.

The no-store-remote directive is not obeyed by the WebAccelerator because the WebAccelerator does not consider itself to be remote from your origin servers.

Overriding HTTP Cache-Control Headers

If ESI Surrogate-Control headers appear in an HTTP response from your origin servers, the WebAccelerator ignores any HTTP 1.1 Cache-Control headers that also appear in that response. For example, if a response includes an ESI max-age directive and an HTTP 1.1 no-cache or no-store header, the WebAccelerator caches the response. The HTTP 1.1 Cache-Control headers supported by the WebAccelerator are described in "HTTP/1.1 Cache-Control Headers" on page 133.

The exception to this rule is that you can use a lifetime policy to cause the WebAccelerator to ignore the ESI max-age directive. For information on setting lifetime policies, see "Defining Lifetime Policies" on page 137.

Surrogate Targeting

Surrogate targeting is an ESI mechanism that provides for control of a specific, identified surrogate device. It is required because web surrogates such as the WebAccelerator might be installed in an environment with other web surrogates. Surrogate targeting allows you to identify header information that is specifically meant for the WebAccelerator.

Surrogate-Control directives can have a parameter that targets the directive to a specific surrogate device. This allows an origin server to specify one directive value for the WebAccelerator, for example, and a different value for all other surrogates.

The parameter is appended to the directive using a semicolon (;). A comma (,) separates the parameter from any other directives that follow. For example:

```
Surrogate-Control: max-age=3600+600;pivia,max-age=3600
```

This example is parsed as:

`max-age=3600+600;pivia`	resulting in a max-age of 30 minutes with a 10 minute stand-in interval for the WebAccelerator.
`max-age=3600`	resulting in a max-age of 30 minutes for all other devices.

The device token in this example is `pivia`. The `pivia` device token was defined to the origin servers in the ESI Surrogate-Capability request header that the WebAccelerator added to the request before it proxied the request to the origin servers.

The WebAccelerator always uses a device token of `pivia` in the ESI Surrogate-Capability header. For more information on the ESI Surrogate-Capability header, see "Processing an ESI Response" on page 98.

X-PvControl Response Header

The X-PvControl response header provides a powerful way for you to change the way a response is processed. Generally, when a request is assigned to a node in the Request Type Hierarchy, the response generated for the request is also assigned to the same node. This is convenient, because when you define policies for certain types of requests, those policies are applied appropriately to the response. You do not need to differentiate between the request and the response it generates.

However, there can be circumstances in which you want a different set of policies to apply to a response based on the properties of that response. The information in a request might be misleading, seeming to request a page that is not cache-able but when the response comes back from the origin servers, the WebAccelerator might determine that it is cache-able.

You can use the Rewrite Engine to examine a response and insert an X-PvControl response header. The X-PvControl response header defines a different node for the response. Any processing that occurs after the response leaves the Rewrite Engine is governed by the policies associated with that new node, rather than the node that had been assigned to the request. Only certain types of processing are affected by this. See "Response Header Processing" on page 215.

Header Format

The format of the X-PvControl response header is:

```
X-PvControl: response-app-name=fully.qualified.node
```

where *`fully.qualified.node`* is the fully qualified name of the node you want to assign the response to. This includes any applicable grandparent, parent, and child nodes to fully describe the leaf node. For example, to apply any policies you defined for the Search node, which is under the Applications node, to the response, use:

```
X-PvControl: response-app-name=Applications.Search
```

To use the Rewrite Engine to change responses, see Appendix B, Rewrite Engine.

X-PvInfo Response Header

The X-PvInfo response header provides a way for you to assess how your policies are working. The WebAccelerator inserts this header into responses that it sends to clients. The purpose of this header is informational only.

This header includes an `S`*`nnnnn`* code indicating where the response was served from:

S10101	The response was served from WebAccelerator memory cache.
S10102	The response was served from WebAccelerator disk cache.
S10201	This response is the first fetch from an origin server for this content. The content is now in the WebAccelerator disk cache.
S10203	The response is proxied from an origin server because of policy rules.

The header also includes an `A`*`nnn`* code which indicates the node in the Request Type Hierarchy the request was matched to. This enables you to tell which policies were applied to the request. The *`nnn`* portion of the code refers to the ID of the node defined in the development copy of the policy set. This is not the same ID that appears for the node in the production copy. You can view this ID by going to the Policy Editor, clicking on Edit Development to ensure you are looking at the development copy and floating your cursor over the node. An ID appears in the informational pop-up.

The object type and group under which the response is classified is included in the header. Understanding the object type assigned to a response tells you whether any rewrite or compression settings in the assembly policy for node `A`*`nnn`* were overridden.

In the header, the object type is preceded by `OT` and the group is preceded by `OG`. Here is an example of the string:

```
OT/msword,OG/documents
```

The object types and groups defined for your system can be found in the global configuration fragment file:

```
/opt/pivia/dac/policies/cont/globalfragment.xml
```

For more information on how the response was classified as this object type, see “Classifying Responses” on page 213.

Appendix D

Regular Expressions

Strings
Metacharacters
Pattern Definitions

Regular expression support is provided in several places throughout the product. When regular expressions are supported, pattern matching is performed on a whole-string basis. Any regular expression that you provide is assumed to be of the form:

```
^expression$
```

even if you do not explicitly set the beginning of line (^) and end of line ($) indicators.

If you want to perform a substring search, enter the expression using this form:

```
.*expression.*
```

Strings

The string that you match on differs depending on the HTTP data type that you are providing the regular expression for:

HTTP Data Type	String Definition
Host	The value set for the HTTP HOST request header. That is, for: `HOST: www.swanlabs.com` The string that you match on is: `www.swanlabs.com`
Path	The value set for the path portion of the URI. That is, for: `http://www.swanlabs.com/apps/search.jsp?value=computer` the string that you match on is: `/apps/search`
Extension	The value set for the extension portion of the URI. That is, for: `http://www.swanlabs.com/apps/search.jsp?value=computer` the string that you match on is: `jsp`
Query Parameter	The value set for the identified query parameter. That is, if you are matching on the value set for the `action` query parameter, then for: `http://www.swanlabs.com?action=display&PDA` the string that you match on is: `display`
Unnamed Query Parameter	The value set for the unnamed query parameter in the identified ordinal. That is, if you are matching on the value set for the unnamed query parameter in ordinal 2, then for: `http://www.swanlabs.com?action=display&PDA` the string that you match on is: `PDA`

HTTP Data Type	String Definition

Path Segment	The name of the segment key, or the value set for the segment parameter (see "Path Segment" on page 50 for a definition of these terms), depending on what you have identified for the match.
	
	Suppose you identify the path segment in ordinal 1 of the full path as counted from left-to-right. Also suppose you identify parameter ordinal 0. In this case you have identified the segment key, and for this URL:
	
	`http://www.swanlabs.com/apps;AAY34/search.jsp?value=computer`
	
	the string that you match on is:
	
	`apps`
	
	However, if you were to specify parameter ordinal 1, then the string that you match on is:
	
	`AAY34`

Cookie	The value set for the identified cookie. That is, for:
	
	`COOKIE: SESSIONID=TN2MM1QQL`
	
	the string that you match on is:
	
	`TN2MM1QQL`

User Agent	The value set for the HTTP USER_AGENT request header. That is, for:
	
	`USER_AGENT: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)`
	
	the string that you match on is:
	
	`Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)`

Referrer	The value set for the HTTP REFERER request header. That is, for:
	
	`REFERER: http://www.swanlabs.com?action=display`
	
	the string that you match on is:
	
	`http://www.swanlabs.com?action=display`

Header	The value set for the identified header. That is, for:
	
	`Accept-Encoding: gzip`
	
	the string that you match on is:
	
	`gzip`

Metacharacters

These metacharacters are used for pattern matching:
`. ^ $ * + ? [ ] ( ) | \`

If a field expects a regular expression, any time you want to use one of these characters as a literal value, you must escape (\) it.

Pattern Definitions

The table identifies patterns and symbols you use to perform pattern matching.

Pattern	Description
.	Matches any single character.
^	Matches the beginning of the line. Note that the beginning and end of line metacharacters are assumed for every regular expression provided to the WebAccelerator.
$	Matches the end of the line. Note that the beginning and end of line metacharacters are assumed for every regular expression provided to the WebAccelerator.
*	Matches zero or more of the preceding pattern. For example, this expression: `G.*P.*` matches: `GrandPlan` `GreenPeace` `GParse` `GP` You can begin a pattern with this character, which is the same as using: `.*` So these are identical expressions: `*Plan` `.*Plan`

Pattern	Description

+	Matches one or more of the preceding pattern. For example, this expression:
	`G.+P.*`
	matches:
	`GrandPlan`
	`GreenPeace`
	Beginning a pattern with this character, `+Plan` for example, is illegal. Instead, use:
	`.+Plan`

?	Matches zero or one of the preceding pattern. For example, this expression:
	`G.?P.*`
	matches:
	`GParse`
	`GP`
	Beginning a pattern with this character, `?Plan` for example, is illegal. Instead, use:
	`.?Plan`

[...]	Matches a set of characters. You can list the characters in the set using a string made of the characters to match on. For example, this expression:
	`C[AHR]`
	matches:
	`CAT`
	`CHARISMA`
	`CRY`
	You can also provide a range of characters using a dash. For example, this expression:
	`AA[0-9]+`
	matches:
	`AA269812209`
	`AA2`
	However, this is not a match:
	`AAB2`
	To match on any alphanumeric character, both upper and lower case, use:
	`[a-zA-Z0-9]`

Pattern	Description

[^...]	Matches any character not in the set. Just as with [...], you can specify just the individual characters, or a range of characters by using a dash (-). For example, this expression:
	`C[^AHR].*`
	matches:
	`CLEAR`
	`CORY`
	`CURRENT`
	`but not:`
	`CAT`
	`CHARISMA`
	`CRY`

(...)	Matches on the regular expression inside the parenthesis as a group. For example, this expression:
	`AA(12)+CV`
	matches:
	`AA12CV`
	`AA121212CV`

`exp1|exp2`	Matches either *`exp1`* or *`exp2`*, where *`exp1`* and *`exp2`* are regular expressions. For example, this expression:
	`AA([de]12|[zy]13)CV`
	matches:
	`AAd12CV`
	`AAe12CV`
	`AAz12CV`
	`AAy13CV`

Index

Symbols

! 104
- 225
!= 103
$ 224
& 104
* 224
< 103
<!--esi 113
<= 103
<--′ 113
== 103
> 103
>= 103
? 225
? in path 68
[...] 225
[^...] 226
^ 224
| 104, 226
‹...› 226
Ž 224

A

Accelerator
  activating 27
  adding to a cluster 27
  creating a cluster 25
  description 2
  rewrite script process 197
accessing
  Admin Tool 14
  Policy Editor 35
account, invalidation service 153
ACTION ESI invalidation tag 158
Add button 15
Admin Tool
  accessing 13
  Policy Editor 35
advanced invalidation selectors 156
  attributes
    BODYEXP 162
    HOST 161
    METHOD 162
    URIEXP 163
    URIPREFIX 161
  elements 157
    cookie 164
    header 164, 165
    PV_COOKIE_EXP 167
    PV_CUSTOM_HEADER_EXP 170
    PV_EXT_EXP 174
    PV_HEADER_EXP 169
    PV_HOST_ALT 171
    PV_QUERY_EXP 165
    PV_REFERER_EXP 168
    PV_SEGMENT_L_R_EXP 172
    PV_SEGMENT_R_L_EXP 173
    PV_UNAME_QUERY_EXP 166
    PV_USER_AGENT_EXP 168
always proxy rules 122
ambiguous query parameters 77
application
  assigning policies 28
  authorizing invalidation requests 153
  choosing Accelerators 28
  creating a profile 28
application matching
  best fit rule 67
  case sensitivity 66

data types 64
extension 67
path 67
ESI 114
example 68
node priority 67
overriding on a response 218
overview 7
path prefixes 68
policy, See also matching policies 63
precedence rules 66
Request Type Hierarchy 66
unmatched request 67
application policies
for remote 29
managing 35
overview 9
assembly policies 81, 91
creation example 95
defining 91
example 94
inheritance 94
input parameters 81
options 82
overriding 213
Request Type Hierarchy 93
specifying rewrite scripts 198
assembly, overriding rewrite scripts 213
auto logout 14

B

Back buttons 17
back end processing 212
base64-encode 153
basic invalidation selectors 155
best fit rule in application matching 67
BODYEXP advanced selector attribute 162
browser cache
Express Connect 87
Express Loader 84
minimum age 131
browser persistent connections 87

C

cache invalidation
ESI 144, 152
advanced selectors 156
elements 157
basic selectors 155
effective time 158
example of 159
request authorization 153
request format 154
request requirements 152
responses 153
sending requests 152
invalidation rules 144, 193
lifetime 146
when in effect 145, 193
invalidation triggers 144, 147
defining 148
example 150
Request Type Hierarchy 150
source of 149
targets 149
state of 150
manual invalidation 144, 146
configuring 147
example 147
Cache-Control
request header 136
max-age 136
min-fresh 137
no-cache 136
no-store 136
response directives 134
response header 98
max-age 134, 135
no-cache 134
no-store 134
private 134
s-maxage 134, 135
cached objects 3
caching content, requirements 193
caching policies
Admin Tool 13
assembly 91
managing 35

mapping HTTP requests 7
mapping HTTP requests to 66
matching 65
publishing 45
system behavior 47
user interface 35
variation 75
default behavior 73
changing password 17
changing the node on a response 218
classifying responses 213
client IP data type 53
cluster
assigning to an application 28
public address 26
request handling 23
steps to create 25
compiled response
input parameters 81
page assembly instructions 81
time to live 129
compression
assembly policies 88
overriding 45
overriding policies 213
conditional GET 84
connection mapping types 176
connection properties 32
connection timeout, destination host field 33
connections policies
creation example 180
defining 176
example of 179
inheritance 179
multiple parameters 179
Request Type Hierarchy 179
content assembly
ESI 97
gzip-encoding 82, 88
on proxies 89
parameter value substitution 83
parameters 92
content expiration 129
content invalidation 193
content lifetime
browser cache minimum age 131
Cache-Control headers
request 136
response 134
default time to live 139
ESI headers 133
example 139
HTTP Cache-Control headers 133
HTTP lifetime heuristic 132
maximum age 130
mechanisms 132
precedence rule 138
stand-in period 131
Surrogate-Control headers 215
content variation, See also variation policies
default caching behavior 73
example 79
content-length response header 194
content-type request header 192
controlling cache behavior 6
cookie advanced selector element 164
cookie data type 51, 223
customer support 18

D

daylight savings time 17
debugger
accessing 96
content assembly 95
enabling 96
debugging policies 219
default connection type 176
Delete link in UI 16
deploying policies 45
destination host
adding to host map 30
connection timeout field 33
lifetime field 33
read write field 33
retries field 33
security field 33
setting options 32
valid hostname 22
device token 218
directives

Cache-Request 136
Cache-Response 134
supported for Surrogate-Control 216
unsupported 217
DNS 19
do not cache rules 122
domain mapping
creating 30
unmapped 21

E

Edge-Side Include, See also ESI 7
edit host mapping 30
See also domain mapping 30
Edit link in UI 16
ESI
content lifetime headers 215
device token 218
invalidator service password 31
uses 11
ESI block
conditional processing 111
exception handling 109
ESI cache invalidation 144
advanced selectors 156
elements 157
basic selectors 155
effective time 158
example of 159
introduction 12
request authorization 153
request format 154
request requirements 152
responses 153
sending requests 152
specification 152
ESI page assembly
case sensitivity 101
closing tags 101
comments 113
debugger 96
example 115
expressions 101
complex 102
literals 102
negation 102
operand association 102
fragments 97
caching 99
requesting 99
fragments with different lifetimes 100
introduction 11
language definition 101
operators 102
binary 103
comparison 103, 104
logical 104
overview 97
parameter value substitution 114, 115
response caching 98
response processing 98
serving from cache 99
Surrogate-Control 215
Surrogate-Control headers 98
templates 99, 114
variables 104
default values 106
dictionaries 105
lists 105
XML and 101
ESI surrogate targeting 217
ESI template example 117
esi:attempt 109
esi:choose 111
esi:choose block 101
esi:comment 112
esi:except 109
esi:include
caching fragments 99
domain mapping 30
esi:try 109
processing 98
syntax 107
use 97
esi:inline
caching fragments 99
syntax 108
use 97
esi:otherwise 111
esi:remove 112

esi:try 109
esi:vars 107
esi:when 111
examples
    application matching 68
    cache invalidation
        effective time 158
        ESI 159
            basic invalidation selector 155
            invalidation request 153
            invalidation response 154
        manual 147
        triggers 150
    connections 179
    content assembly 94
    content lifetime 139
    content variation 79
    ESI page assembly 115
    ESI template 117
    Express Loader 86, 88
    hit logs 189
    HTML fragment 119
    proxy override rules 125
    proxy rules 124
    Request Type Hierarchy 60
    responses cached 184
expired content 129
expires response header 134, 135
exporting policies 39
Express Connect
    enabling 86
    number of subdomains 34
    options described 34
    specifying options 31
    subdomain prefix 34
Express Loader
    description 84
    example of 86, 88
    expiration time 85
    pv tag 85
    tags supported 85, 87
expressions, regular 221
extension data type 48, 222
    precedence 67

F

figures
    multiple Accelerators 2
    Policy Editor 40
    request decision flow 24
    request/response flow 5
files, global configuration fragment 213
flow of request handling 23
format codes 186
forwarding requests to a proxy server 26
forwarding requests to another cluster 29
fragments 97
    caching 99
    different lifetimes 100
    requesting 99

G

GET method 192
GET, conditional 84
globalfragment.xml file 213
gzip-encoding 82
    overriding 45
    specifying in assembly policies 88

H

HEAD method 192
header
    data type 52
    structured 52
header advanced selector element 164, 165
headers
    response processing 215
    Surrogate-Control 133
    X-PvControl 218
    X-PvInfo 219
Help navigation 17
hit logs
    example of 189
    format codes 186
    formats 186
    tailoring format 185
home page 18
Host Address 32

HOST advanced selector attribute 161
host data type 48, 222
host map
    request handling 23
    steps to create 30
HOST request header 192
Host Type 32
hostname
    specifying 22
    wildcards 22
hostnames, valid 22
HTML fragments 97
    caching 99
    different lifetimes 100
    example of 119
    requesting 99
HTTP 1.1
    Cache-Control headers 133
        ESI Surrogate-Control 133
        ESI Surrogate-Control override 217
HTTP data types
    application matching 64
    assembly
        substitution source 92
        substitution target 93
    Client IP 53
    cookie 51, 223
    definition of 45
    extension 48, 222
    header 52
    host 48, 222
    invalidation triggers 149
    method 49
    path 48, 222
    path segment 50, 223
    protocol 48
    proxying 122
    query parameter 49, 222
    referrer 51, 223
    significant for content 74
    unnamed query parameter 49, 222
    user agent 51, 223
HTTP LAST_MODIFIED response header 132
HTTP lifetime header 138
HTTP lifetime heuristic 132
HTTP protocol, supported 192
HTTP request headers 192
    Cache-Control: max-age 136
    Cache-Control: min-fresh 137
    Cache-Control: no-cache 136
    Cache-Control: no-store 136
    content-type 192
    expect 192
    pragma: no-cache 136
    size limits 192
HTTP requests 192
    mapping to caching policies 7, 66
    redirected 192
    requirements for servicing 192
    size limits 192
    supported data types 45
    unmatched 67
HTTP response codes 183
HTTP response headers
    Cache-Control: max-age 134, 135
    Cache-Control: no-cache 134
    Cache-Control: no-store 134
    Cache-Control: private 134
    Cache-Control: s-maxage 134, 135
    content-length 194
    expires 134, 135
    last-modified 135
    pragma: no-cache 134
    transfer-encoding 193
HTTP responses 181
    requirements for caching 193
HTTP_ACCEPT_LANGUAGE 104
HTTP_COOKIE 104
HTTP_HOST 104
HTTP_REFERER 105
HTTP_USER_AGENT 105

I

icon
    inherited 42
    overridden inheritance 43
importing policies 39
inheritance 57
    icon showing inheritance 42
    icon showing inheritance overridden 43

overriding 59
invalidation effective time 158
invalidation rules 193
definition of 144
lifetime 146
when in effect 145, 193
invalidation selectors, advanced 156
invalidation service account 153
invalidation triggers 147
defining 148
example 150
Request Type Hierarchy 150
source of 149
data types 149
targets 149
state of 150
invalidator service password 31
IP address of client 53
italicized links 18

L

last-modified response header 135
leaf node
creating 61
inheritance 57
priority 56
lifetime policies
creation example 140, 141
defining 137
inheritance 138
Request Type Hierarchy 138
stand-in period 131
lifetime, destination host field 33
lifetime, See also content lifetime 130
linearizing PDFs
enabling 90
rewrite script 205
log mover 185
logout 14

M

Management Console, description 2
markup, debugger 95
matching policies
creation example 69
defining 65
parameters 65
max-age
esi:inline 109
request header 136
response header 134, 135
maximum age 130
metacharacters 224
method
required in request header 192
supported types 192
METHOD advanced selector attribute 162
method data type 49
min-fresh request header 137
minimum age 131

N

navigating Help 17
no-cache
request header 136
response header 134
nodes
creating 61
reassigning a node 218
no-store
request header 136
response header 134
no-store-remote directive 217
NTLM
caching 182
default connection type 177

O

object type, rewrite script 198
One-to-One connection type 176
origin servers
properties 32
using Surrogate-Control response header 215

P

parameter
  header 52
  structured 52
parameter value substitution 82, 83
  assembly policies 92
  ESI and 114, 115
  random values 84
  source 92
  target 93
  target URL 93
password
  changing 17
  invalidation service 153
  invalidator service 31
path data type 48, 222
  precedence 67
path segment 50
  data type 50, 223
  segment keys 50
  segment ordinals 50
  segment parameters 50
pathname prefixes 68
PDF files, linearize 205
pdf.code script 205
Performance Reports 10
persistent connections 87
pivia device token 218
Pivia, now Swan Labs 2
policies
  affected by response headers 212
  assembly 91
  assessing 219
  assigning to an application 28
  changing policies for a response 218
  connections 176
    types 176
  inherit overridden icon 43
  inherited icon 42
  lifetime 137
  matching 65
  policy set 36
  proxying 123, 193
  publishing 45
  responses cached 182
    authorized content 182
    content assurance 182
    response codes 183
  system behavior 47
  used by remote clusters 29
  variation 75
Policy Editor 35
policy set
  copying 38
  creating 38
  definition 36
  deleting 39
  exporting 39
  importing 39
  pre-defined 37
  renaming 38
post data maximum size 192
POST method 192
pragma: no-cache
  request header 136
  response header 134
precedence rule, application matching 66
prefix
  path names 68
  subdomain 88
  URL 92
private
  cache control directive 134
  response header 134
processing
  response headers 215
  responses 212
product documentation 18
protocol
  required in request header 192
  supported types 192
protocol data type 48
proxy for fresh content 129
  content assembly on 89
proxy override rules 121, 125
  creation example 125, 126
  defining 125
proxy rules 123
  always proxy 122
  cache conditions 192
proxy server

connection properties 32
defining in host map 32
forwarding requests 26
proxying policies
creation example 124
data types 122
defining 123
inheritance 127
no cache 193
parameters 123
Request Type Hierarchy 126
publishing policies 45
pv tag 85
PV_ABS_REMOVALTTL 158
PV_COOKIE_EXP 167
PV_CUSTOM_HEADER_EXP 170
PV_EXT_EXP 174
PV_HEADER_EXP 169
PV_HOST_ALT 171
PV_NO_CACHE_PERIOD 158
PV_QUERY_EXP 165
PV_REFERER_EXP 168
PV_SEGMENT_L_R_EXP 172
PV_SEGMENT_R_L_EXP 173
PV_UNAME_QUERY_EXP 166
PV_USER_AGENT_EXP 168

Q

query parameter data type 49, 222
variation policies 77
QUERY_STRING 105

R

random value substitution 84
read write, destination host field 33
referrer data type 51, 223
refreshing content 129
regular expressions 221
metacharacters 224
remote policies 29
REMOVALTTL 158
Remove link in UI 16
reporting problems 18
request directives 136
Request Type Hierarchy
application matching 66
assembly policies 93
connection policies 179
example 60
inheritance 57
overriding 59
lifetime policies 138
managing 56
node creation 61
node priority 56
overview 55
proxying policies 126
responses cached policies 183
tree structure 55
variation policies 78
request-based connection type 177
managing 177
value groups 178
Requested Host
adding to host map 30
description 21
wildcards in hostname 22
requested hosts, unmapped 21
requests
incoming 23
policies used when forwarding 29
to unmapped hosts
policies used when forwarding 26
requirements
caching responses 193
servicing client requests 192
response codes
403 21
404 131
response directives 134
response header
X-PvControl 218
X-PvInfo 219
response processing 212
assigning types 213
responses
changing effective policies 218
classifying 213
completeness 182

completeness requirements 193
specifying ESI 133
responses cached policies
authorized content 182
content assurance 182
creation example 184
defining 182
example of 184
inheritance 183
Request Type Hierarchy 183
response codes 183
restrictions
post data size 192
size limit on requests 192
retries, destination host field 33
Rewrite Engine
Accelerator rewrite process 197
cl.code script 205
common.code script 205
content.code script 205
expressions 199
functions, built-in 200
functions, predefined 205
headers.code script 205
language definition 195
language literals 200
language processing 198
linearize PDFs 205
overriding assembly policy settings 213
pdf.code script 205
procedures 199
request_get mnemonics 206
specifying scripts 90
transform utility 206
X-PvControl response header 218

S

security issues with unmapped hosts 22
Security, destination host field 33
segment keys 50
segment ordinals 50
segment parameter ordinals 50
segment parameters 50
session timeout 14
session_expires_time 14
s-maxage response header 134, 135
source 149
specifying hostnames 22
stand-in period 131
in max-age directive 216
structured parameters in request headers 52
supported
methods 192
protocols 192
surrogate targeting 217
Surrogate-Control headers, content lifetime 215
Swan Labs WebAccelerator
formerly Pivia Performance Server 2

T

technical support 18
templates 99
time to live 129
caching considerations 193
default value 139
system limits 138
time zone, setting 17
transfer-encoding response header 193
transform utility 206
applying to responses 214
TTL, See also time to live 129

U

unmapped hosts 21
changing settings 26
security concerns 22
unmatched requests
during application matching 67
unnamed query parameter data type 49, 222
variation policies 77
URI, required in request header 192
URIEXP advanced selector attribute 163
URIPREFIX advanced selector attribute 161
user agent data type 51, 223
user interface, accessing 13

V

valid hostnames 22
variation policies
- ambiguous query parameters 77
- creation example 79, 80
- defining 75
- inheritance 78
- multiple matching parameters 77
- parameters 75
- Request Type Hierarchy 78
- servicing requests, effect on 78
- value groups 76
- when optional 74
- when required 73

View link in UI 16

W

web browser cache
- age 131
- Express Connect 87
- Express Loader 84

web browser persistent connections 87
wildcards in hostname 22

X

XML 101
X-PvControl response header 218
- format 218

X-PvInfo response header 219