Blue Coat Systems™
Proxy*SG* Content Policy Language Guide

Content Policy Language Guide

Blue Coat Systems Inc.
650 Almanor Avenue
Sunnyvale, California 94086
Technical Support
info@bluecoat.com

(408) 220-2200 Voice
(408) 220-2250 FAX
(866) 302-2628
(866) 362-2628
www.bluecoat.com

Printed in U.S.A.

Document Number: 231-02586

Document Revision: 3.1.2

THIRD PARTY COPYRIGHT NOTICES

Blue Coat Systems, Inc. Security Gateway Operating System (SGOS) version 3 utilizes third party software from various sources. Portions of this software are copyrighted by their respective owners as indicated in the copyright notices below.

The following lists the copyright notices for:

BPF

Copyright (c) 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996

The Regents of the University of California. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that: (1) source code distributions retain the above copyright notice and this paragraph in its entirety, (2) distributions including binary code include the above copyright notice and this paragraph in its entirety in the documentation or other materials provided with the distribution, and (3) all advertising materials mentioning features or use of this software display the following acknowledgement:

This product includes software developed by the University of California, Lawrence Berkeley Laboratory and its contributors.

Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

DES

Software DES functions written 12 Dec 1986 by Phil Karn, KA9Q; large sections adapted from the 1977 public-domain program by Jim Gillogly.

EXPAT

Copyright (c) 1998, 1999, 2000 Thai Open Source Software Center Ltd.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Finjan Software

Copyright (c) 2003 Finjan Software, Inc. All rights reserved.

Flowerfire

Copyright (c) 1996-2002 Greg Ferrar

ISODE

ISODE 8.0 NOTICE

Acquisition, use, and distribution of this module and related materials are subject to the restrictions of a license agreement. Consult the Preface in the User's Manual for the full terms of this agreement.

4BSD/ISODE SMP NOTICE

Acquisition, use, and distribution of this module and related materials are subject to the restrictions given in the file SMP-READ-ME.

UNIX is a registered trademark in the US and other countries, licensed exclusively through X/Open Company Ltd.

MD5

RSA Data Security, Inc. MD5 Message-Digest Algorithm

Copyright (c) 1991-2, RSA Data Security, Inc. Created 1991. All rights reserved.

License to copy and use this software is granted provided that it is identified as the "RSA Data Security, Inc. MD5 Message-Digest Algorithm" in all material mentioning or referencing this software or this function.

License is also granted to make and use derivative works provided that such works are identified as "derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm" in all material mentioning or referencing the derived work.

RSA Data Security, Inc. makes no representations concerning either the merchantability of this software or the suitability of this software for any particular purpose. It is provided "as is" without express or implied warranty of any kind.

THE BEER-WARE LICENSE" (Revision 42):

<phk@FreeBSD.org <mailto:phk@FreeBSD.org>> wrote this file. As long as you retain this notice you can do whatever you want with this stuff. If we meet some day, and you think this stuff is worth it, you can buy me a beer in return. Poul-Henning Kamp

Microsoft Windows Media Streaming

Copyright (c) 2003 Microsoft Corporation. All rights reserved.

OpenLDAP

Copyright (c) 1999-2001 The OpenLDAP Foundation, Redwood City, California, USA. All Rights Reserved. Permission to copy and distribute verbatim copies of this document is granted.

http://www.openldap.org/software/release/license.html

The OpenLDAP Public License Version 2.7, 7 September 2001

Redistribution and use of this software and associated documentation ("Software"), with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain copyright statements and notices,

2. Redistributions in binary form must reproduce applicable copyright statements and notices, this list of conditions, and the following disclaimer in the documentation and/or other materials provided with the distribution, and

3. Redistributions must contain a verbatim copy of this document.

The OpenLDAP Foundation may revise this license from time to time. Each revision is distinguished by a version number. You may use this Software under terms of this license revision or under the terms of any subsequent revision of the license.

THIS SOFTWARE IS PROVIDED BY THE OPENLDAP FOUNDATION AND ITS CONTRIBUTORS ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OPENLDAP FOUNDATION, ITS CONTRIBUTORS, OR THE AUTHOR(S) OR OWNER(S) OF THE SOFTWARE BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The names of the authors and copyright holders must not be used in advertising or otherwise to promote the sale, use or other dealing in this Software without specific, written prior permission. Title to copyright in this Software shall at all times remain with copyright holders.

OpenLDAP is a registered trademark of the OpenLDAP Foundation.

OpenSSH

Copyright (c) 1995 Tatu Ylonen <ylo@cs.hut.fi>, Espoo, Finland. All rights reserved

This file is part of the OpenSSH software.

The licences which components of this software fall under are as follows. First, we will summarize and say that all components are under a BSD licence, or a licence more free than that.

OpenSSH contains no GPL code.

1) As far as I am concerned, the code I have written for this software can be used freely for any purpose. Any derived versions of this software must be clearly marked as such, and if the derived work is incompatible with the protocol description in the RFC file, it must be called by a name other than "ssh" or "Secure Shell".

[Tatu continues]

However, I am not implying to give any licenses to any patents or copyrights held by third parties, and the software includes parts that are not under my direct control. As far as I know, all included source code is used in accordance with the relevant license agreements and can be used freely for any purpose (the GNU license being the most restrictive); see below for details.

[However, none of that term is relevant at this point in time. All of these restrictively licenced software components which he talks about have been removed from OpenSSH, i.e.,

- RSA is no longer included, found in the OpenSSL library

- IDEA is no longer included, its use is deprecated

- DES is now external, in the OpenSSL library

- GMP is no longer used, and instead we call BN code from OpenSSL

- Zlib is now external, in a library

- The make-ssh-known-hosts script is no longer included

- TSS has been removed

- MD5 is now external, in the OpenSSL library

- RC4 support has been replaced with ARC4 support from OpenSSL

- Blowfish is now external, in the OpenSSL library

[The licence continues]

Note that any information and cryptographic algorithms used in this software are publicly available on the Internet and at any major bookstore, scientific library, and patent office worldwide. More information can be found e.g. at "http://www.cs.hut.fi/crypto".

The legal status of this program is some combination of all these permissions and restrictions. Use only at your own responsibility. You will be responsible for any legal consequences yourself; I am not making any claims whether possessing or using this is legal or not in your country, and I am not taking any responsibility on your behalf.

NO WARRANTY

BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR

A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

2) The 32-bit CRC compensation attack detector in deattack.c was contributed by CORE SDI S.A. under a BSD-style license.

Cryptographic attack detector for ssh - source code

Copyright (c) 1998 CORE SDI S.A., Buenos Aires, Argentina. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that this copyright notice is retained. THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES ARE DISCLAIMED. IN NO EVENT SHALL CORE SDI S.A. BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY OR CONSEQUENTIAL DAMAGES RESULTING FROM THE USE OR MISUSE OF THIS SOFTWARE.

Ariel Futoransky <futo@core-sdi.com> <http://www.core-sdi.com>

3) ssh-keygen was contributed by David Mazieres under a BSD-style license.

Copyright 1995, 1996 by David Mazieres <dm@lcs.mit.edu>. Modification and redistribution in source and binary forms is permitted provided that due credit is given to the author and the OpenBSD project by leaving this copyright notice intact.

4) The Rijndael implementation by Vincent Rijmen, Antoon Bosselaers and Paulo Barreto is in the public domain and distributed with the following license:

@version 3.0 (December 2000)

Optimised ANSI C code for the Rijndael cipher (now AES)

@author Vincent Rijmen <vincent.rijmen@esat.kuleuven.ac.be>

@author Antoon Bosselaers <antoon.bosselaers@esat.kuleuven.ac.be>

@author Paulo Barreto <paulo.barreto@terra.com.br>

This code is hereby placed in the public domain.

THIS SOFTWARE IS PROVIDED BY THE AUTHORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

5) One component of the ssh source code is under a 3-clause BSD license, held by the University of California, since we pulled these parts from original Berkeley code.

Copyright (c) 1983, 1990, 1992, 1993, 1995

The Regents of the University of California. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

6) Remaining components of the software are provided under a standard 2-term BSD licence with the following names as copyright holders:

Markus Friedl
Theo de Raadt
Niels Provos
Dug Song
Aaron Campbell
Damien Miller
Kevin Steves
Daniel Kouril
Wesley Griffin
Per Allansson
Nils Nordman
Simon Wilkinson

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

OpenSSL

Copyright (c) 1995-1998 Eric Young (eay@cryptsoft.com). All rights reserved.

http://www.openssl.org/about/

http://www.openssl.org/about/

OpenSSL is based on the excellent SSLeay library developed by Eric A. Young <mailto:eay@cryptsoft.com> and Tim J. Hudson <mailto:tjh@cryptsoft.com>.

The OpenSSL toolkit is licensed under a Apache-style license which basically means that you are free to get and use it for commercial and non-commercial purposes.

This package is an SSL implementation written by Eric Young (eay@cryptsoft.com). The implementation was written so as to conform with Netscapes SSL.

This library is free for commercial and non-commercial use as long as the following conditions are adhered to. The following conditions apply to all code found in this distribution, be it the RC4, RSA, lhash, DES, etc., code; not just the SSL code. The SSL documentation included with this distribution is covered by the same copyright terms except that the holder is Tim Hudson (tjh@cryptsoft.com).

Copyright remains Eric Young's, and as such any Copyright notices in the code are not to be removed. If this package is used in a product, Eric Young should be given attribution as the author of the parts of the library used. This can be in the form of a textual message at program startup or in documentation (online or textual) provided with the package.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. All advertising materials mentioning features or use of this software must display the following acknowledgement: "This product includes cryptographic software written by Eric Young (eay@cryptsoft.com)" The word 'cryptographic' can be left out if the routines from the library being used are not cryptographic related :-).

4. If you include any Windows specific code (or a derivative thereof) from the apps directory (application code) you must include an acknowledgement: "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"

THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The license and distribution terms for any publicly available version or derivative of this code cannot be changed. i.e. this code cannot simply be copied and put under another distribution license [including the GNU Public License.]

Copyright (c) 1998-2002 The OpenSSL Project. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. All advertising materials mentioning features or use of this software must display the following acknowledgment:

"This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (http://www.openssl.org/)"

4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact openssl-core@openssl.org.

5. Products derived from this software may not be called "OpenSSL" nor may "OpenSSL" appear in their names without prior written permission of the OpenSSL Project.

6. Redistributions of any form whatsoever must retain the following acknowledgment: "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/)"

THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This product includes cryptographic software written by Eric Young (eay@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com).

PCRE

Copyright (c) 1997-2001 University of Cambridge

University of Cambridge Computing Service, Cambridge, England. Phone: +44 1223 334714.

Written by: Philip Hazel <ph10@cam.ac.uk>

Permission is granted to anyone to use this software for any purpose on any computer system, and to redistribute it freely, subject to the following restrictions:

1. This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

2. Regular expression support is provided by the PCRE library package, which is open source software, written by Philip Hazel, and copyright by the University of Cambridge, England.

ftp://ftp.csx.cam.ac.uk/pub/software/programming/pcre/

PHAOS SSLava and SSLavaThin

Copyright (c) 1996-2003 Phaos Technology Corporation. All Rights Reserved.

The software contains commercially valuable proprietary products of Phaos which have been secretly developed by Phaos, the design and development of which have involved expenditure of substantial amounts of money and the use of skilled development experts over substantial periods of time. The software and any portions or copies thereof shall at all times remain the property of Phaos.

PHAOS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE SOFTWARE, OR ITS USE AND OPERATION ALONE OR IN COMBINATION WITH ANY OTHER SOFTWARE.

PHAOS SHALL NOT BE LIABLE TO THE OTHER OR ANY OTHER PERSON CLAIMING DAMAGES AS A RESULT OF THE USE OF ANY PRODUCT OR SOFTWARE FOR ANY DAMAGES WHATSOEVER. IN NO EVENT WILL PHAOS BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBLITY OF SUCH DAMAGES.

RealSystem

The RealNetworks® RealProxy™ Server is included under license from RealNetworks, Inc. Copyright 1996-1999, RealNetworks, Inc. All rights reserved.

SNMP

Copyright (C) 1992-2001 by SNMP Research, Incorporated.

This software is furnished under a license and may be used and copied only in accordance with the terms of such license and with the inclusion of the above copyright notice. This software or any other copies thereof may not be provided or otherwise made available to any other person. No title to and ownership of the software is hereby transferred. The information in this software is subject to change without notice and should not be construed as a commitment by SNMP Research, Incorporated.

Restricted Rights Legend:

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013; subparagraphs (c)(4) and (d) of the Commercial Computer Software-Restricted Rights Clause, FAR 52.227-19; and in similar clauses in the NASA FAR Supplement and other corresponding governmental regulations.

PROPRIETARY NOTICE

This software is an unpublished work subject to a confidentiality agreement and is protected by copyright and trade secret law. Unauthorized copying, redistribution or other use of this work is prohibited. The above notice of copyright on this source code product does not indicate any actual or intended publication of such source code.

STLport

Copyright (c) 1999, 2000 Boris Fomitchev

This material is provided "as is", with absolutely no warranty expressed or implied. Any use is at your own risk.
Permission to use or copy this software for any purpose is hereby granted without fee, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is granted, provided the above notices are retained, and a notice that the code was modified is included with the above copyright notice.

The code has been modified.

Copyright (c) 1994 Hewlett-Packard Company

Copyright (c) 1996-1999 Silicon Graphics Computer Systems, Inc.

Copyright (c) 1997 Moscow Center for SPARC Technology

Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. Hewlett-Packard Company makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. Silicon Graphics makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting

documentation. Moscow Center for SPARC Technology makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

SmartFilter

Copyright (c) 2003 Secure Computing Corporation. All rights reserved.

SurfControl

Copyright (c) 2003 SurfControl, Inc. All rights reserved.

Symantec AntiVirus Scan Engine

Copyright (c) 2003 Symantec Corporation. All rights reserved.

TCPIP

Some of the files in this project were derived from the 4.X BSD (Berkeley Software Distribution) source.

Their copyright header follows:

Copyright (c) 1982, 1986, 1988, 1990, 1993, 1994, 1995

The Regents of the University of California. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. All advertising materials mentioning features or use of this software must display the following acknowledgement:

This product includes software developed by the University of California, Berkeley and its contributors.

4. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Trend Micro

Copyright (c) 1989-2003 Trend Micro, Inc. All rights reserved.

zlib

Copyright (c) 2003 by the Open Source Initiative

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Preface: *Introducing the Content Policy Language*

The *Content Policy Language* (CPL) is a powerful, flexible language that enables you to specify a variety of Web-access policies. Proxy*SG* policy is written in CPL, and every Web request is evaluated based on the installed policy. The language is designed so that policies can be customized to an organization's specific set of users and unique enforcement needs.

CPL uses the settings created when you configured the Proxy*SG* to your specifications.

CPL has the following capabilities:

- Fine-grained control over various aspects of Proxy*SG* behavior.
- Layered policy, allowing for multiple policy decisions for each request.
- Multiple actions triggered by a particular condition.
- Flexibility of user-defined conditions and actions.
- Convenience of predefined common actions and transformations.
- Authentication-aware policy, including user and group configuration.
- Support for multiple authentication realms.
- Configurable policy event logging.
- Built-in debugging.

About the Document Organization

This document is organized for easy reference, and is divided into the following sections and chapters:

Table 2.1: Manual Organization

Chapter 1 – *Overview of Content Policy Language*	This chapter provides an overview of CPL, including concepts, CPL basics, writing and troubleshooting policy and upgrade/downgrade issues.
Chapter 2 – *Managing CPL*	Building upon Chapter 1, this chapter discusses understanding transactions, timing, layers, and sections, defining policies, and best practices.
Chapter 3 – *Conditions*	This reference guide contains the list of conditions that are supported by CPL and provides an explanation for the usage.
Chapter 4 – *Properties*	This reference guide contains the list of properties that are supported by CPL and provides an explanation for the usage.
Chapter 5 – *Actions*	This reference guide contains the list of actions that are supported by CPL and provides an explanation for the usage.
Chapter 6 – *Definitions*	This reference guide contains the list of definitions that are supported by CPL and provides an explanation for the usage.
Appendix A – *Glossary*	Terms used in this manual are defined in this appendix.
Appendix B – *Troubleshooting*	Using policy trace properties is explained in this appendix.
Appendix C – *Recognized HTTP Headers*	This appendix lists all recognized HTTP 1.1 headers and indicates how the Proxy*SG* interacts with them.

Table 2.1: Manual Organization (Continued)

Appendix D – *CPL Substitutions*	This appendix lists all substitution variables available in CPL.
Appendix E – *Filter File Syntax*	This appendix provides a summary of the syntax and evaluation order used in CacheOS version 4.*x* filter files.
Appendix F – *Upgrading from CacheOS 4.x*	If you upgrade from CacheOS 4.x, you need to be aware of the concerns and issues that affect a policy upgrade to SGOS 3.x.

Supported Browsers

The ProxySG Management Console supports Microsoft® Internet Explorer 5 and 6, and Netscape® Communicator 4.78, 6.2, and 7.1.

The Management Console uses the Java Runtime Environment. All browsers come with a default, built-in JRE, and you should use this default JRE rather than an independent JRE version downloaded from Sun® Microsystems.

Related Blue Coat Documentation

Blue Coat 6000 and 7000 Installation Guide

Blue Coat 400 Series Installation Guide

Blue Coat 800 Series Installation Guidel

Proxy*SG Command Line Interface Reference*

Document Conventions

The following section lists the typographical and Command Line Interface (CLI) syntax conventions used in this manual.

Table 2.2: Typographic Conventions

Conventions	Definition
Italics	The first use of a new or Blue Coat-proprietary term.
`Courier font`	Command line text that appears on your administrator workstation.
`Courier Italics`	A command line variable that is to be substituted with a literal name or value pertaining to the appropriate facet of your network system.
`Courier Boldface`	A Proxy*SG* literal to be entered as shown.
`{ }`	One of the parameters enclosed within the braces must be supplied
`[ ]`	An optional parameter or parameters.
`|`	Either the parameter before or after the pipe character can or must be selected, but not both. To more clearly indicate that only one can be chosen, no spaces are put between the pipe and the options.

Contents

Preface: Introducing the Content Policy Language

About the Document Organization ix
Supported Browsers ix
Related Blue Coat Documentation x
Document Conventions x

Chapter 1: Overview of Content Policy Language

Concepts 19
Transactions 19
Policy Model 20
Role of CPL 21
CPL Language Basics 21
Comments 21
Rules 21
Notes 22
Quoting 23
Layers 24
Sections 24
Definitions 25
Referential Integrity 26
Substitutions 27
Writing Policy Using CPL 27
Authentication and Denial 28
Installing Policy 29
CPL General Use Characters and Formatting 29
Troubleshooting Policy 30
Upgrade/Downgrade Issues 30
CPL Syntax Deprecations 30
Conditional Compilation 31

Chapter 2: Managing Content Policy Language

Understanding Transactions and Timing 33
Administrator Transactions 33
Proxy Transactions 33
Cache Transactions 35
Forwarding Transactions 36
Timing 36
Understanding Layers 37
<Admin> Layers 37
<Cache> Layers 38
<Exception> Layers 39

<Forward> Layers 39
<Proxy> Layers 40
Layer Guards 40
Timing 41
Understanding Sections 41
[Rule] 42
..... [url]43
[url.domain] 43
[url.regex] 43
[server_url.domain] 43
Section Guards 44
Defining Policies 44
Blacklists and Whitelists 45
General Rules and Exceptions to a General Rule 45
Best Practices 48

Chapter 3: Condition Reference

Condition Syntax 49
Pattern Types 50
Unavailable Triggers 51
Layer Type Restrictions 51
Global Restrictions 51
Condition Reference 51
acl= 52
admin.access= 53
attribute.name= 54
authenticated= 56
bitrate= 57
category= 59
client.address= 60
client.protocol= 61
condition= 62
console_access= 64
content_admin= 65
content_management 66
date[.utc]= 67
day= 68
exception.id= 69
ftp.method= 71
group= 72
has_attribute.name= 74
has_client= 76
hour= 77

http.method= 79
http.request.version= 80
http.response.code= 81
http.response.version= 82
http.transparent_authentication= 83
http.x_method= 84
im.buddy_id= 85
im.chat_room.conference= 86
im.chat_room.id= 87
im.chat_room.invite_only= 88
im.chat_room.type= 89
im.chat_room.member= 90
im.chat_room.voice_enabled= 91
im.file.extension= 92
im.file.name= 93
im.file.path= 94
im.file.size= 95
im.message.opcode= 96
im.message.route= 97
im.message.size= 98
im.message.text= 99
im.message.type= 100
im.method= 101
im.user_id= 102
live= 103
method= 104
minute= 106
month= 107
protocol= 108
proxy.address= 109
proxy.card= 110
proxy.port= 111
realm= 112
release.id= 114
release.version= 115
request.header.header_name= 116
request.header.header_name.address= 117
request.header.Referer.url= 118
request.x_header.header_name= 121
request.x_header.header_name.address= 122
response.header.header_name= 123
response.x_header.header_name= 124

server_url= 125
socks= 128
socks.accelerated= 129
socks.method= 130
socks.version= 131
streaming.client= 132
streaming.content= 133
time= 134
tunneled= 136
url= 137
user= 144
user.domain= 146
user.x509.issuer= 147
user.x509.serialNumber= 148
user.x509.subject= 149
weekday= 150
year= 151

Chapter 4: Property Reference

Property Reference 153
access_log( ) 154
access_server( ) 155
action( ) 156
advertisement( ) 157
allow 158
always_verify( ) 159
authenticate( ) 160
authenticate.force( ) 162
authenticate.mode( ) 163
authenticate.use_url_cookie( ) 165
block_category( ) 166
bypass_cache( ) 167
cache( ) 168
check_authorization( ) 170
content_filter_override( ) 171
cookie_sensitive( ) 172
delete_on_abandonment( ) 173
deny( ) 174
deny.unauthorized( ) 175
direct( ) 176
dynamic_bypass( ) 177
exception( ) 178
exception.autopad( ) 179

force_cache( ) 180
force_deny( ) 181
force_exception( ) 182
force_patience_page( ) 183
forward( ) 184
forward.fail_open( ) 185
ftp.server_connection( ) 186
ftp.server_data( ) 187
ftp.transport( ) 188
http.force_ntlm_for_server_auth( ) 189
http.request.version( ) 190
http.response.version( ) 191
icp( ) 192
im.strip_attachments( ) 193
integrate_new_hosts( ) 194
label( ) 195
log.rewrite.field-id( ) 196
log.suppress.field-id( ) 197
max_bitrate( ) 198
never_refresh_before_expiry( ) 199
never_serve_after_expiry( ) 200
patience_page( ) 201
pipeline( ) 202
prefetch( ) 203
reflect_ip( ) 204
reflect_vip( ) 205
refresh( ) 206
remove_IMS_from_GET( ) 207
remove_PNC_from_GET( ) 208
remove_reload_from_IE_GET( ) 209
request.filter_service( ) 210
request.icap_service( ) 212
response.icap_service( ) 213
service( ) 214
socks.accelerate( ) 215
socks.authenticate( ) 216
socks.authenticate.force( ) 217
socks_gateway( ) 218
socks_gateway.fail_open( ) 219
streaming.transport( ) 220
terminate_connection( ) 221
trace.destination( ) 222

trace.request( ) 223
trace.rules( ) 224
ttl( ) 225
ua_sensitive( ) 226

Chapter 5: Action Reference

Argument Syntax 227
Action Reference 227
append( ) 228
delete( ) 229
delete_matching( ) 230
im.alert( ) 231
log_message( ) 232
notify_email( ) 233
notify_snmp( ) 234
redirect( ) 235
replace( ) 236
rewrite( ) 237
set( ) 240
transform 242
virus_check( ) 244

Chapter 6: Definition Reference

Definition Names 245
define action 246
define active_content 248
define category 250
define condition 252
define domain condition 254
define javascript 255
define prefix condition 257
define server_url.domain condition 258
define subnet 260
define url condition 261
define url.domain condition 263
define url_rewrite 265
restrict dns 267
restrict rdns 268
transform active_content 269
transform url_rewrite 270

Appendix A: Glossary

Appendix B: Testing and Troubleshooting
Enabling Rule Tracing 275
Enabling Request Tracing 276
Using Trace Information to Improve Policies 276

Appendix C: Recognized HTTP Headers

Appendix D: CPL Substitutions

Appendix E: Filter File Syntax
Filter File Overview 299
Filter File Structure 299
Filter-Part Components 300
Action-Part Components 305
Evaluation Order 306

Appendix F: Upgrading from CacheOS

Index

Chapter 1: *Overview of Content Policy Language*

The Content Policy Language (CPL) is a programming language with its own concepts and rules that you must follow.

This chapter provides an overview of CPL, including the following topics:

- "Concepts"
- "CPL Language Basics"
- "Writing Policy Using CPL"
- "Troubleshooting Policy"
- "Upgrade/Downgrade Issues"

Concepts

The term *policy*, as used here, refers to configuration values and rules applied to render decisions on authentication requirements, access rights, quality of service, or content transformations (including rewrites and off-box services that should be used to process the request or response). Often, the policy references system configuration for the default values for some settings and then evaluates rules to see if those settings should be overridden.

CPL is a language for specifying the policy rules for the Proxy*SG*. Primarily, it controls the following:

- User Authentication requirements
- Access to Web-related resources
- Cache content
- Various aspects of request and response processing
- Access logging

You can create policy rules using either the Visual Policy Manager (VPM), which is accessible through the Management Console, or by composing CPL.

Before reading sample CPL or trying to express your own policies in CPL, Blue Coat recommends that you understand the fundamental concepts underlying policy enforcement in the Proxy*SG* appliances. This section provides an overview of important concepts.

Transactions

In the CPL context, a *transaction* is the encapsulation of a request for service and any associated response for the purposes of policy evaluation and enforcement. In most cases, a transaction is created for each unique request for service, and the transaction exists for the time taken to process the request and deliver the response.

The transaction serves the following purposes:

- Exposes request and response state for testing during policy evaluation.

This provides the ability to test various aspects of a request, such as the IP address of the client and the URL used, or the response, such as the contents of any HTTP headers.

- Ensures policy integrity during processing.

  The lifetime of a transaction may be relatively long, especially if a large object is being fetched over slow networks and subjected to off-box processing services such as content filtering and virus scanning. During this time, changes to configuration or policy rules may occur, which would result in altering the policy decisions that affect a transaction. If a request was evaluated against one version of policy, and some time later the associated response were evaluated against a different version of policy, the outcome would be unpredictable and possibly inconsistent.

  The transaction ensures that both the request and the response are evaluated against the version of policy that was current when the transaction was created. To ensure that new policy is respected, long lived transactions such as those involved in streaming, or large file downloads, are re-evaluated under new policy. Re-evaluation applies to both the request and response, and any resulting new decisions that cannot be honoured (such as new authentication requirements) result in transaction termination.
- Maintains policy decisions relevant to request and response processing.
- Various types of transactions are used to support the different policy evaluation requirements of the individual protocols: administrator, cache, and proxy transactions.
- In a few special cases, two or more transactions can be created for a single request. For example, if an HTTP request is made via the SOCKS proxy (on port 1080 of the ProxySG), then it is possible for two transactions to be created: a SOCKS proxy transaction, and an HTTP proxy transaction. You can see these transactions for yourself if you turn on policy tracing. A new entry is added to the policy trace file for each transaction.

Policy Model

Each transaction begins with a default set of decisions, many of which are taken from configuration of the system. These defaults include such things as forwarding hosts or SOCKS gateways. The most important default decision affects whether or not requests should be allowed or denied. The defaults for the various transaction types are:

- Administrator Transaction— the default is to deny requests.

  By default, administration is only available through one of the methods that bypasses policy evaluation. These are:

  - accessing the CLI through the serial console
  - accessing the CLI through RSA authenticated SSH
  - logging into the Management Console or CLI using the console credentials

  Specific rights must be granted through policy to enable other administration methods.
- Cache Transactions—the default is to allow requests.

  These requests originate from the ProxySG itself, and are used primarily to maintain the state of content. Additional policy can be added to specifically deny requests for specific content, and to distinguish content management requests from other cache transactions.
- Proxy Transactions—the default is taken from system configuration.

For new Proxy*SG* appliances, the default is to deny all requests. For Proxy*SG* appliances being upgraded from 4.x, the default is to allow all requests. In either case, the Proxy*SG* can be configured for either default. The default setting is displayed in policy listings.

The proper approach to writing `<proxy>` layer policy depends on whether or not the default is to allow or deny requests. The default proxy policy is configurable and represents the starting point for writing policy to control proxy transactions. The default proxy policy is reported at the top of every policy listing generated by the Proxy*SG*.

```
; Default proxy policy is DENY
```

That line in a policy listing is a CPL comment, defining the starting point for proxy policy.

Role of CPL

CPL is the language used to express policy that depends on the runtime evaluation of each transaction. Policy is written in CPL, installed on the Proxy*SG*, and is evaluated during request processing to override any default decisions taken from configuration.

CPL Language Basics

The following sections provide an overview of the CPL language. In order to concentrate on higher level themes, CPL elements are informally introduced and discussed. Detailed specifications for each of these elements is left to the reference portion of this manual.

Comments

Any line starting with ';' is a comment.

A semicolon (;) following a space or tab introduces a comment that extends to the end of the line (except where the semicolon appears inside quotes as part of a trigger pattern expression or property setting).

For example:

```
; This is a comment.
```

Comments can appear anywhere in policy.

Rules

A policy *rule* consists of a *condition* and some number of *property* settings, written in any order. Rules are generally written on a single line, but can be split across lines using a special line continuation character. When a rule is evaluated, the condition is tested for that particular transaction. If the condition evaluates to True, then all of the listed property settings are executed and evaluation of the current layer ends. The rule is said to *match*. If the condition evaluates to False for that transaction, it is said to *miss*.

In turn, a condition is a boolean combination of *trigger* expressions. Triggers are individual tests that can be made against components of the request (`url=`), response (`response.header.Content-Type=`), related user (`user=`, `group=`), or system state (`time=`).

With a few notable exceptions, triggers test one aspect of request, response, or associated state against a boolean expression of values.

For the conditions in a rule, each of the triggers is logically *anded* together. In other words, the condition is only true if each one of the trigger expressions is true.

Properties are settings that control transaction processing, such as deny, or the handling of the object, such as cache(no), indicating that the object is not to be cached locally. At the beginning of a transaction, all properties are set to their default values. As the policy is evaluated in sequence, rules that match might set a property to a particular value. A property retains the final value setting when evaluation ends, and the transaction is processed accordingly. Properties that are not set within the policy maintain their default values.

The logical form of a policy rule could be expressed as:

```
if condition is true then set all listed properties as specified
```

The following is an example of a simple policy rule:

```
url.domain=example.com time=0900..1700 exception(policy_denied)
```

It states that the `exception( )` property is set to `policy_denied` if both of the following triggers test true:

- The request is made for a page from the domain `example.com`
- The request is made between 9 a.m. and 5 p.m.

Notes

- CPL triggers have the form *`trigger_name=pattern_expression`*
- CPL properties have the form *`property_name(setting)`*, except for a few imperative gestures such as `allow` and `deny`.
- The text in policy rules is case-insensitive, with a few exceptions identified in the following chapters.
- Policy listings are normalized in several ways. First, condition and action definitions which may appear anywhere in the source, will be grouped following the policy rules. Second, the order of the conditions and properties on a rule may change, since the CPL compiler always puts a deny or allow at the beginning of the rule, and orders conditions to optimize evaluation. Finally, several phrases are synonyms for phrases that are preferred. In the output of `show policy`, the preferred form is listed instead of the synonym.

  Four such synonyms are:

  - `exception(authorization_failed)`, which is a synonym for the preferred `deny.unauthorized`
  - `force_exception(authorization_failed)`, which is a synonym for the preferred `force_deny.unauthorized`
  - `exception(policy_denied)`, which is a synonym for the preferred `deny`
  - `exception(no)`, which is a synonym for the preferred `allow`.

- More complex boolean expressions are allowed for the *pattern_expression* in the triggers. For example, the second part of the condition in the simple rule shown above could be "the request is made between 9 a.m. and noon or between 1 p.m. and 5 p.m", expressed as:

  ```
  ... time=(0900..1200 || 1300..1700) ...
  ```

  Boolean expression are built from the specific values allowed with the trigger, and the boolean operators `!` (not), `&&` (and), `||` (or) and `()` for grouping. More details are found in the Trigger Reference chapter. Alternative values may also be separated by a comma—this is often more readable than using the '`||`' operator. For example, the following rule will deny service to requests for pages in either one of the two domains listed.

  ```
  url.domain=(example.com, another.com) deny
  ```

- Long lines can be split using '`\`' as a line continuation character. The '`\`' must be the last character on the line and be preceded by space or Tab. For example:

  ```
  url.domain=example.com time=0900..1700 \
      deny
  ```

  Do not use a semicolon to add comments within such a continued line: everything following the semicolon, including text on the continued lines, will be treated as part of the comment. For example:

  ```
  url.domain=example.com \ ; missplaced comment
      deny
  ```

  becomes

  ```
  url.domain=example.com ; missplaced comment deny
  ```

  In other words, the effect was to continue the comment.

Quoting

Certain characters are considered *special* by CPL and have meaning as punctuation elements of the language. For example = (equal) separates a trigger name from its associated value, and blank space separates expressions in a rule. To use a value that contains one of these characters, the value must be quoted with either single (') or double (") quotation marks, so that the special characters are not interpreted as punctuation. Text within single quotation marks can include any character other than a single quotation mark. Text within double quotation marks can include any character other than a double quotation mark. Here are some examples of where quoting is necessary:

```
user="John Doe" ; value contains a space

url="www.example.com/script.cgi?param=value"  ; value contains '='

deny( "You don't have access to that page!" ) ; several special chars
```

The full list of characters that should be quoted when they appear can be found in the reference manual. Note that you can quote any string in CPL without affecting interpretation, even if the quotes are not strictly needed. For convenience, you can quote any value that consists of more than letters and/or numbers.

```
user="john.doe" ; quotes not required, but can be used
```

Important: Within a `define action` or `define url_rewrite` statement, you must use double quotes ("), not single quotes (') to delimit a string.

Layers

A policy *layer* is a CPL construct used to evaluate a set of rules and reach one decision. Separating decisions helps control policy complexity, and is done through writing each decision in a separate layer. Each layer has the form:

```
<layer_type [label]> [layer_condition][layer_properties] ...
   layer_content
```

where:

- The *layer_type* defines the transactions evaluated against this policy, and restricts the triggers and properties allowed in the rules used in the layer. See the following Layer Types section.
- The optional *label*, separated from the layer type by space, is a CPL User-defined Identifier (see section Chapter 2), basically an alphabetic character followed by alphanumeric or underscore characters.
- The optional *layer_condition* is a list of triggers, all of which must evaluate to true before the layer content is evaluated.
- The optional *layer_properties* is a list of properties that will become the default settings for those properties for any rule matched in the layer. These can be overridden by explicitly setting a different value for that property in a specific rule within the layer.
- The *layer_content* is a list of rules, possibly organized in *sections*. (see following). A layer must contain at least one rule.

Collectively, the layer_condition and layer_properties are often referred to as a *layer guard* expression.

If a rule has the logical form "if (condition is true) then set properties", a layer has the form:

```
if (layer_condition is true) then
  {
    if (rule1_condition is true) then
       set layer_properties then set rule1 properties
    else if (rule2_condition is true) then
       set layer_properties then set rule2 properties
    else if (rule3_condition is true) then
       set layer_properties then set rule3 properties
    ...
  }
```

Within a layer, the first rule that matches terminates evaluation of that layer.

Layers within a policy are evaluated from top to bottom, with rules in later layers taking precedence over rules in earlier layers.

In CPL, all policy rules are written in a layer. A rule cannot appear in policy preceding any layer header.

Sections

The rules in layers can optionally be organized in one or more sections, which is a way of grouping rules together. A *section* consists of a section header followed by a list of rules.

A section has the form:

```
[section_type  [label]] [section_condition][section_properties]
   section_content
```

where:

- The *section_type* defines the syntax of the rules used in the section, and the evaluation strategy used to evaluate those rules. The square brackets [ ] surrounding the section name (and optional label) are required.
- The optional *label,* separated from the section type by space, is a CPL User-defined Identifier similar to a layer label.
- The optional *section_condition* is a list of triggers, all of which must evaluate to true before the section content is evaluated.
- The optional *section_properties* is a list of properties that will become the default settings for those properties for any rule matched in the section. These override any layer property defaults and can in turn be overridden by explicitly setting a different value for that property in a rule within the section.
- The *section_content* is a list of rules. A section must contain at least one rule.

Collectively, the section_condition and section_properties are often referred to as a *section guard* expression.

A layer with sections has the logical form:

```
if (layer_condition is true) then
  {
    if (section1_condition is true then
      {
        if (rule1A_condition is true) then
            set layer_properties then section_properties then rule1A properties
        else if (rule1B_condition is true) then
            set layer_properties then section_properties then set rule1B
properties
        ....
      }
    else if (section2_condition is true then
      {
        if (rule2A_condition is true) then
            set layer_properties then section_properties then rule2A properties
       else ...
      }
    ...
  }
```

Definitions

Two types of definitions are used in CPL:

- *Named definitions* that are explicitly referenced by policy
- *Anonymous definitions* that apply to all policy evaluation and are not referenced directly in rules.

Named Definitions

There are various types of named definitions. Each definition is given a user defined name that is then used in rules to refer to the definition. This section highlights a few of the definition types, as an overview of the topic. Refer to the Definitions reference chapter for more details.

Subnet Definitions

Subnet definitions are used to define a list of IP addresses or IP subnet masks that can be used to test any of the IP addresses associated with the transaction, for example, the client's address or the request's destination address.

Condition Definitions

Condition definitions can include any triggers that are legal in the layer referencing the condition. The `condition=` trigger is the exception to the rule that triggers can test only one aspect of a transaction. Since conditions definitions can include other triggers, `condition=` triggers can test multiple parts of the transaction state. Also, condition definitions allow for arbitrary boolean combinations of trigger expressions.

Category Definitions

Category definitions are used to extend vendor content categories or to create your own. These categories are tested (along with any vendor defined categories) using the `category=` trigger.

Action Definitions

An action takes arguments and is wrapped in a named action definition block. Actions are turned on or off for a transaction through setting the `action( )` property. The action property has syntax that allows for individual actions to be turned on and off independently. When the action definition is turned on, any actions it contains operate on their respective arguments.

Transformer Definitions

A transformer definition is a kind of named definition that specifies a transformation that is to be applied to an HTTP response. There are three types: `url_rewrite` definitions, `active_content` definitions, and `javascript` definitions.

Anonymous Definitions

Two types of anonymous definitions modify policy evaluation, but are not referenced by any rules. These definitions serve to restrict DNS and Reverse-DNS lookups and are useful in installations where access to DNS or Reverse-DNS resolution is limited or problematic.

Referential Integrity

Policy references many objects defined in system configuration, such as authentication realms, forward hosts, SOCKS gateways, and the like. CPL enforces the integrity of those references by ensuring that the entities named in policy exist and have appropriate characteristics at the time the policy is compiled. During runtime, any attempts to remove a configured object that is referenced by currently active policy will fail.

To remove a configured entity, such as a realm, that is referenced by policy, new policy must be installed with all references to that realm removed. New transactions will open against a version of

policy that does not require the realm. Once all outstanding transactions that required reference to the realm have completed, the realm can be removed from configuration.

Substitutions

The actions used to rewrite the URL request or to modify HTTP request headers or HTTP response headers often need to reference the values of various elements of the transaction state when constructing the new URL or header value. CPL provides support for various substitutions, which will expand at runtime to the indicated transaction value. Substitutions have the form:

`$(`*`name`*`)`

For example, the substitution `$(user)` expands to the authenticated user name associated with the transaction. If policy did not require that user to authenticate, the substitution expands to an empty string.

Substitutions can also be used directly in the values specified to some CPL properties, such as when setting text in a message that will be displayed to users.

Substitutions are available for a variety of purposes. For a categorized list of the substitutions available, see Appendix D: "CPL Substitutions".

Writing Policy Using CPL

A policy file is the unit of integration used to assemble policy.

Policy written in CPL is stored in one of four files on the Proxy*SG*. These files are the following:

- VPM: This file is reserved for use by the Visual Policy Manager.
- Local: When the VPM is not being used, the Local file will typically contain the majority of the policy rules for a system. When the VPM is being used, this file might be empty, it might include rules for advanced policy features that are not available in the VPM, or it might otherwise supplement VPM policy.
- Central: This file is typically managed by Blue Coat, although you can have the Proxy*SG* point to a custom Central policy file instead.
- Forward: The Forward policy file is normally used for all Forward policy, although you can use it to supplement any policy created in the other three policy files. The Forward policy file will contain Advanced Forwarding rules when the system is upgraded from a previous version of SGOS (2.x) or CacheOS (4.x).

Each of the files may contain rules and definitions, but an empty file is also legal. (An empty file specifies no policy and has no effect on the Proxy*SG*.)

Cross file references are allowed but the definitions must be installed before the references, and references must be removed before definitions are removed.

The final installed policy is assembled from the policy stored in the four files by concatenating their contents. The order of assembly of the VPM, Central and Local policy files is configurable. The recommended evaluation order is VPM, Local, Central. The Forward policy file is always last.

Authentication and Denial

One of the most important timing relationships to be aware of is the relation between authentication and denial. Denial can be done either before or after authentication, and different organizations have different requirements. For example, suppose an organization requires the following:

- Protection from denial of service attacks by refusing traffic from any source other than the corporate subnet.
- The user name of corporate users is to be displayed in access logs, even when the user request has been denied.

The following example demonstrates how to choose the correct CPL properties. First, the following is a sample policy that is not quite correct:

```
define subnet corporate_subnet
   10.10.12.0/24
end

<Proxy>
   client.address=!corporate_subnet deny ; filter out strangers
   authenticate(MyRealm) ; this has lower precedence than deny

<Proxy>
   ; user names will NOT be displayed in the access log for the denied requests
   category=Gambling exception(content_filter_denied)
```

In this policy, requests coming from outside the corporate subnet are denied, while users inside the corporate subnet are asked to authenticate.

Content categories are determined from the request URL and can be determined before authentication. Deny has precedence over authentication, so this policy denies the user request before the user is challenged to authenticate. Therefore, the user name is not available for access logging. Note that the precedence relation between deny and authenticate does not depend on the order of the layers, so changing the layer order will not affect the outcome.

The CPL property `force_authenticate()`, however, has higher precedence than deny, so the following amended policy ensures that the user name is displayed in the access logs:

```
define subnet corporate_subnet
   10.10.12.0/24
   end

<Proxy>
   client.address=!corporate_subnet deny ; filter out strangers
   force_authenticate(MyRealm) ; this has higher precedence than deny

<Proxy>
   ; user names will be displayed in the access log for the denied requests
   category=Gambling exception(content_filter_denied)
```

The timing for authentication over the SOCKS protocol is different. If you are using the SOCKS authentication mechanism, the challenge is issued when the connection is established, so user identities are available before the request is received, and the following policy would be correct.

```
define subnet corporate_subnet
   10.10.12.0/24
end
```

```
<Proxy>
   client.address=!corporate_subnet deny ; filter out strangers
   socks.authenticate(MyRealm) ; this happens earlier than the category test

<Proxy>
   ; user names be displayed in the access log for the denied requests
   category=Gambling exception(content_filter_denied)
```

Note that this only works for SOCKS authenticated users.

Installing Policy

Policy is installed by installing one of the four policy files (VPM, Local, Central or Forward). Installing one new file causes the most recent versions of the other three files to be loaded, the contents concatenated in the order specified by the current configuration, and the resulting complete policy compiled.

If any compilation errors are detected, the new policy file is not installed and the policy in effect is unchanged.

Refer to Chapter 12, "Advanced Policy," of the Proxy*SG Configuration and Management Guide* for specific instructions on installing a policy file.

CPL General Use Characters and Formatting

The following characters and formatting have significance within policy files in general, outside of the arguments used in condition expressions, the values used in property statements, and the arguments used in actions.

Character	Example	Significance
Semicolon (`;`)	`; Comment` `<Proxy> ; Comment`	Used either inline or at the beginning of a line to introduce text to be ignored during policy evaluation. Commonly used to provide comments.
Newline	`deny server_url.scheme=mms deny server_url.domain=xyz.com`	CPL expects most constructs (layers, sections, rules, definitions) to begin on a new line. When not preceded by a line continuation character, a newline terminates a layer header, section header, the current rule, clause within a defined condition, or action within an action definition.
Line Continuation	`\`	A line continuation character indicates that the current line is part of the previous line.
Whitespace	`< proxy >` `weekday = ( 3 || 7 ) deny`	Used to enhance readability. Whitespace can be inserted between tokens, as shown in this example, without affecting processing. In addition, quoted strings can include whitespace. However, numeric ranges, such as `weekday = 1..7`, cannot contain whitespace.
Angle brackets (`< >`)	`<Proxy>`	Used to mark layer headings.
Square brackets (`[ ]`)	`[Rule]`	Used to mark section names.

Equal sign (=)	`server_url.scheme=mms`	Used to indicate the value a condition is to test.
Parentheses ( )	`service(no)`	Used to enclose the value that a property is to be set to, or group components of a test.

Troubleshooting Policy

When installed policy does not behave as expected, use policy tracing to understand the behavior of the installed policy.

Tracing records additional information about a transaction and re-evaluates the transaction when it is terminated; however, it does not show the timing of evaluations through transaction processing. The extra processing required significantly impacts performance, so do not enable tracing in production environments unless you need to reproduce and diagnose a problem. If tracing is used on a system in production, attempt to restrict which transactions are traced. For example, you can trace only requests from a test workstation by defining the tracing rules as conditional on a client.address= trigger that tests for that workstation's IP address.

For more information on generating and retrieving policy trace, see Appendix B: "Testing and Troubleshooting".

While policy traces can show the rule evaluation behavior, they do not show the final effect of policy actions like HTTP header or URL modifications. To see the result of these policy actions it is often useful to actually view the packets sent and received. The PCAP facility can be used in conjunction with tracing to see the effect of the actions set by the matching rules.

Upgrade/Downgrade Issues

Specific upgrade downgrade issues will be mentioned in the release notes accompanying your version of SGOS. This section highlights general upgrade downgrade issues related to policy written in CPL.

CPL Syntax Deprecations

As the power of CPL has increased, the CPL language has evolved. To allow continuous evolution, the CPL language constructs are now more regular and flexible. Older language constructs have been replaced with new constructs of equal or greater power.

However, this also implies that support for old language constructs will eventually be dropped to help maintain the runtime efficiency of evaluation. As part of the migration strategy, the CPL compilation warnings might include warnings regarding the use of deprecated constructs. This class of warning is special, and indicates use of a CPL language element that will not be supported in the next major release of SGOS. Eliminate deprecation warnings by migrating the policy identified by the warning to more modern syntax, which is usually indicated in the warning message. Attempts to upgrade to the next major release might fail, or result in a failure to load policy, unless all deprecation warnings are eliminated.

Conditional Compilation

Occasionally, you might be required to maintain policy that can be applied to appliances running different versions of SGOS and requiring different CPL. CPL provides the following conditional compilation directive that tests the SGOS version (such as 2.1.06):

```
release.version= <version number range>
```

The range is a standard CPL range test: *min*`..`*max*, where both minimum and maximum are optional.

The *min* and *max* can be *MAJOR*`.`*MINOR*`.`*DOT*`.`*PATCH*, with *MINOR*, *DOT* and *PATCH* optional. Therefore, rules containing grammar introduced in 2.1.07 can be protected with

```
#if release.version=2.1.07..
; guarded rules
...
#endif
```

while grammar introduced in 2.2 can be protected with:

```
#if release.version=2.2..
; guarded rules
...
#endif
```

Chapter 2: *Managing Content Policy Language*

As discussed in Chapter 1, Content Policy Language policies are composed of transactions that are placed into rules and tested against various conditions.

This chapter discusses the following:

- "Understanding Transactions and Timing"
- "Understanding Layers"
- "Understanding Sections"
- "Defining Policies"
- "Best Practices"

Understanding Transactions and Timing

Transactions are classified as administrator, proxy, cache, and forwarding. Only a subset of layer types, conditions, properties, and actions is appropriate for each of these four transaction types.

Administrator Transactions

An administrator transaction evaluates policy in `<Admin>` layers. The policy is evaluated in two stages:

- Before the authentication challenge.
- After the authentication challenge.

If an administrative user logs in to the Proxy*SG* Management Console, and the administrator's Web browser is proxied through that same Proxy*SG*, then a proxy transaction is created and `<Proxy>` policy is evaluated *before* the administrator transaction is created and `<Admin>` policy is evaluated. In this case, it is possible for an administrator to be denied access to the Management Console by proxy policy.

Important: Policy is not evaluated for serial console access, RSA authenticated SSH access, managers logged in using the console account credentials, or SNMP traffic.

Proxy Transactions

When a client connects to one of the proxy service ports configured on the secure proxy appliance (refer to Chapter 6: "Proxies" of the *Configuration and Management Guide*), a proxy transaction is created to cover both the request and its associated response.

A proxy transaction evaluates policy in `<Proxy>`, `<Cache>`, `<Forward>` and `<Exception>` layers. The `<Forward>` layers are only evaluated if the transaction reaches the stage of contacting an origin server to satisfy the request (this is not the case if the request is satisfied by data served from cache, or if the transaction is terminated by an exception). The `<Exception>` layers are only evaluated if the transaction is terminated by an exception.

Each of the protocol-specific proxy transactions has specific information that can be tested—information that may not be available from or relevant to other protocols. HTTP Headers and Instant Messaging buddy names are two examples of protocol-specific information.

Other key differentiators among the proxy transaction subtypes are the order in which information becomes available and when specific actions must be taken, as dictated by the individual protocols. Variation inherent in the individual protocols determines *timing*, or the sequence of evaluations that occurs as the transaction is processed.

The following table summarizes the policy evaluation order for each of the protocol-specific proxy transactions.

Table 2.1: When Policy is Evaluated

Transaction Type	Policy is Evaluated....
Tunneled TCP transactions	before the connection is established to the origin server.
HTTP proxy transactions	Before the authentication challenge.
	After the authentication challenge, but before the requested object is fetched.
	Before making an upstream connection, if necessary.
	After the object is fetched
FTP over HTTP transactions:	Before the authentication challenge.
	After the authentication challenge, but before the required FTP commands are executed.
	Before making an upstream connection, if necessary.
	After the object is fetched.
Transparent FTP transactions	Policy is examined before the requested object is fetched.
Real Media streaming transactions	Before the authentication challenge.
	After the authentication challenge, but before getting object information.
	Before making an upstream connection, if necessary.
	After the object information is available, but before streaming begins.
	After streaming begins (this evaluation can be done multiple times, for example after playback is paused and restarted).
Windows Media MMS streaming transactions	Before the authentication challenge.
	Before making an upstream connection, if necessary.
	After the authentication challenge but before getting object information.
	After the object information is available, but before streaming begins.
	After streaming begins (this evaluation can be done multiple times, for example after playback is paused and restarted).

Table 2.1: When Policy is Evaluated (Continued)

Windows Media HTTP streaming transactions	Before the authentication challenge.
	After the authentication challenge, but before the requested object is fetched.
	Before making an upstream connection, if necessary. (Up to this point it is similar to an HTTP transaction.) What happens at this stage depends on negotiations with the origin server: • After the origin server is contacted, if the User Agent header denotes the Windows Media player and the server supports Microsoft streaming HTTP extensions, it finishes like an MMS transaction: Object information is available at this stage but streaming has not begun. • If the User-Agent header is not a Windows Media player or the server does not support Microsoft streaming extensions, it finishes like an HTTP transaction: The requested object is fetched, and policy is evaluated

Some conditions cannot be evaluated during the first stage; for example, the user and group information will not be known until stage two. Likewise, the response headers and MIME type are unavailable for testing until stage three. For conditions, this is known as the *earliest available time*.

Policy decisions can have similar timing considerations, but this is known as the *latest commit time*. In this example, the requirement to authenticate must be known at stage one, and a forwarding host or gateway must be determined by stage three.

Cache Transactions

Cache transactions are initiated by the Proxy*SG* in order to load or maintain content in the local object store during adaptive refresh or pipelining, or as a result of a `content distribute` CLI command. These may be HTTP, FTP, or streaming media transactions. Since no specific user is associated with these transactions, content related policy is evaluated for cache transactions, but user related policy is not evaluated.

A cache transaction evaluates policy in `<Cache>` and `<Forward>` layers. The `<Forward>` layers are only evaluated if an origin server must be contacted to complete the transaction.

The following is a list of cache transactions:

- A content distribute transaction that is initiated by the `content distribute` CLI command. A content distribute transaction may use one of the following protocols: HTTP, HTTPS, Real Media, or Windows Media. This type of transaction may be preceded by a separate Administrator transaction, since the administrator must be authenticated and authorized to use the command.
- Pipeline transactions (HTTP only).
- Advertisement transactions (HTTP only).
- If-modified-since transactions (HTTP only).
- Refresh transactions (HTTP only).
- ICP transactions.

Cache transactions have no client identity since they are generated internally by the Proxy*SG*, and they do not support authentication or authorization. Therefore, they do not support conditions such as `client.address=` and `group=`, or the `authenticate()` property.

An HTTP cache transaction is examined in two stages:

- Before the object is retrieved from the origin server.
- After the object is retrieved.

Forwarding Transactions

A forwarding transaction is created when the Proxy*SG* needs to evaluate forwarding policy before accessing a remote host and no proxy or cache transaction is associated with this activity. Examples include sending a heart-beat message, and downloading an installable list from an HTTP server.

A forwarding transaction only evaluates policy in `<Forward>` layers.

Timing

As stated in the discussion of proxy transactions, various portions of the transaction information become available at different points in the evaluation, and each protocol has specific requirements for when each decision must be made. The CPL triggers and properties are designed so that wherever possible, the policy writer is shielded from the variations among protocols by making the timing requirements imposed by the CPL accommodate all the protocols. Where this is not possible (because using the most restrictive timing causes significant loss of functionality for the other protocols), protocol specific triggers have been introduced. When evaluated against other protocols, these triggers return the `not applicable` value or `N/A`. This results in the rule being skipped (the expression evaluates to false, no matter what it is). It is possible to explicitly guard such rules so that they are only evaluated against appropriate transactions.

The variation in trigger and property timings implies that within a policy rule a conflict is possible between a condition that can only be tested relatively late in the evaluation sequence and a property that must be set relatively early in the evaluation sequence. Such a rule results in a compile-time error.

For example, here is a rule that would be incorrect for evaluating any transaction:

If the user is in group `xyz`, require authentication.

The rule is incorrect because group membership can only be determined after authentication and the rule tests group membership and specifies the authentication realm, a property that must be set before the authentication challenge can be issued. The following code illustrates this incorrect rule and the resulting message at compile time:

```
group=xyz authenticate(MyRealm)

Error: Late condition guards early action: 'authenticate(MyRealm)'
```

It is, however, correct for the authentication requirement to be conditional on the client address (`client.address=`) or proxy port (`proxy.port=`), as these can be determined at the time the client connection is established and therefore are available from the beginning of a proxy transaction.

For the HTTP protocol, `authenticate()` can be conditional on the URL (`url=`), but for MMS streaming, only the Host portion of the URL can be tested (`url.host=`). Recall the outline of the evaluation model for Windows Media transactions presented in "Understanding Transactions and Timing" on page 33.

As another example, consider the following:

```
response.header.Content-type=”text/html” forward( somehost )
```

But policy cannot determine the value of the Content-type response header until the response is returned. The Proxy*SG* cannot contact the server to get the response until policy determines what hosts or gateways to route through to get there. In other words, policy must set the `forward()` property. But policy cannot commit the forwarding action until the Content-type response header has been determined. Again, since the condition is not testable until later in the request (after the time at which the property must be set), an error is received.

Understanding Layers

Five types of layers are allowed in any policy file. The layer type determines the kinds of transaction its rules will act upon. The token used in the header identifies the layer type.

- `<Admin>`—Used to define policy that controls access to the management console and the command line. Policy is not evaluated for serial console access or SNMP traffic, however.
- `<Cache>`—Used to list policy rules that are evaluated during both cache and proxy transactions.
- `<Exception>`—Exception layers are evaluated when a proxy transaction is terminated by an exception.
- `<Forward>`—Forward layers are only evaluated when the current transaction requires an upstream connection. Forwarding policy is generally distinct and independent of other policies, and is often used as part of maintaining network topologies.
- `<Proxy>`—Used to list policy rules that are evaluated during a proxy transaction.

Important: Only a subset of the conditions, properties, and actions available in the policy language is permitted within each layer type; the remainder generate compile-time errors. The CPL Reference for the conditions, properties, and actions describes where they can be used.

<Admin> Layers

`<Admin>` layers hold policy that is executed by Administrator transactions. This policy is used to specify an authentication realm; to allow or deny administrative access based on the client's IP address, credentials, and type of administrator access requesuested (read or write); and to perform any additional logging for administrative access.

Important: When traffic is explicitly proxied, it arrives at the `<Admin>` layer with the client IP address set to the Proxy*SG*'s IP address; therefore, the `client.address=` condition is not useful for explicitly proxied traffic.

The syntax is:

```
<Admin [label]> [admin_condition][admin_properties] ...
   admin_content
```

where:

- The *<Admin>* layer defines the transactions evaluated against this policy, and restricts the triggers and properties allowed in the rules used in the layer.
- The optional *label*, separated from the layer type by space, is a CPL User-defined Identifier.
- The optional *admin_condition* is a list of triggers, all of which must evaluate to true before the layer content is evaluated. For more information on using conditions, see Chapter 3: "Condition Reference". See also the following Layer Guards section.

- The optional *admin_properties* is a list of properties set if any of the rules in the layer match. These act as defaults, and can be overridden by property settings in specific rules in the layer. For more information on using properties, see Chapter 4: "Property Reference". See also the following Layer Guards section.

<Cache> Layers

<Cache> layers hold policy that is executed by both cache and proxy transactions. Since cache transactions have no concept of a client, all <Cache> layer policy is clientless, so you cannot test client identity using client.address=, user=, group=, and the like.

Certain types of policy must be applied consistently to both proxy and cache transactions to preserve cache consistency. Such policy must not be conditional on client identity or time of day, and belongs in a <Cache> layer. Examples include the following:

- Response virus scanning.
- Cache control policy (other than bypass_cache).
- Modifications to request headers, if the modification affects the content returned by the web server, and the content is cached.
- Rewrites of the request URL that modify the server URL but not the cache URL. (Place rewrites of the request URL that change the cache and server URL to the same value in a <Proxy> layer.)

Only the following properties are safe to make conditional on time or client identity in a <Cache> layer:

- Pipelining
- Tracing, logging
- Freshness checks
- Redirection
- Content transforms

The syntax is:

```
<Cache [label]> [cache_condition][cache_properties] ...
   cache_content
```

where:

- The <*Cache*> layer defines the transactions evaluated against this policy, and restricts the triggers and properties allowed in the rules used in the layer.
- The optional *label*, separated from the layer type by space, is a CPL User-defined Identifier.
- The optional *cache_condition* is a list of triggers, all of which must evaluate to true before the layer content is evaluated. For more information on using conditions, see Chapter 3: "Condition Reference". See also the following Layer Guards section.
- The optional *cache_properties* is a list of properties set if any of the rules in the layer match. These act as defaults, and can be overridden by property settings in specific rules in the layer. For more information on using properties, see Chapter 4: "Property Reference". See also the following Layer Guards section.

<Exception> Layers

`<Exception>` layers are evaluated when a proxy transaction is terminated by an exception. This could be caused by a bad request (for example, the request URL names a non-existent server) or by setting the `deny` or `exception()` properties in policy. Policy in an exception layer can be used to control how access logging is performed for exceptions, such as `authentication_failed`. It can also be used to modify the HTTP response headers in the exception page that is sent to the client.

The syntax is:

```
<Exception [label]> [exception_condition][exception_properties] ...
   exception_content
```

where:

- The *<Exception>* layer defines the transactions evaluated against this policy, and restricts the triggers and properties allowed in the rules used in the layer.
- The optional *label,* separated from the layer type by space, is a CPL User-defined Identifier.
- The optional *exception_condition* is a list of triggers, all of which must evaluate to true before the layer content is evaluated. For more information on using conditions, see Chapter 3: "Condition Reference". See also the following Layer Guards section.
- The optional *exception_properties* is a list of properties set if any of the rules in the layer match. These act as defaults, and can be overridden by property settings in specific rules in the layer. For more information on using properties, see Chapter 4: "Property Reference". See also the following Layer Guards section.

<Forward> Layers

`<Forward>` layers are evaluated when the current transaction requires an upstream connection (and only then: forward policy will not be evaluated for a cache hit). `<Forward>` layers use the `server_url=` tests rather than the `url=` tests so that they are guaranteed to honor any policy that rewrites the URL.

The syntax is:

```
<Forward [label]> [forward_condition][forward_properties] ...
   forward_content
```

where:

- The *<Forward>* layer defines the transactions evaluated against this policy, and restricts the triggers and properties allowed in the rules used in the layer.
- The optional *label,* separated from the layer type by space, is a CPL User-defined Identifier.
- The optional *forward_condition* is a list of triggers, all of which must evaluate to true before the layer content is evaluated. For more information on using conditions, see Chapter 3: "Condition Reference". See also the following Layer Guards section.
- The optional *forward_properties* is a list of properties set if any of the rules in the layer match. These act as defaults, and can be overridden by property settings in specific rules in the layer. For more information on using properties, see Chapter 4: "Property Reference". See also the following Layer Guards section.

<Proxy> Layers

<Proxy> layers define policy for authenticating and authorizing users' requests for service over one of the configured proxy service ports (refer to Chapter 6:"Managing Port Services" in the Proxy*SG Configuration and Management Guide*.). Proxy layer policy involves both both client identity and content. Only proxy transactions are evaluated against <Proxy> layers.

The syntax is:

```
<Proxy [label]> [proxy_condition][proxy_properties] ...
   proxy_content
```

where:

- The <*Proxy*> layer defines the transactions evaluated against this policy, and restricts the triggers and properties allowed in the rules used in the layer.
- The optional *label*, separated from the layer type by space, is a CPL User-defined Identifier.
- The optional *proxy_condition* is a list of triggers, all of which must evaluate to true before the layer content is evaluated. For more information on using conditions, see Chapter 3: "Condition Reference". See also the following Layer Guards section.
- The optional *proxy_properties* is a list of properties set if any of the rules in the layer match. These act as defaults, and can be overridden by property settings in specific rules in the layer. For more information on using properties, see Chapter 4: "Property Reference". See also the following Layer Guards section.

Layer Guards

Often, the same set of conditions or properties appears in every rule in a layer. For example, a specific user group for which a number of individual cases exist where some things are denied:

```
<Proxy>
   group=general_staff url.domain=competitor.com/jobs deny
   group=general_staff url.host=bad_host deny
   group=general_staff condition=whatever deny
   ; etc.
   group=general_staff allow
```

You can factor out the common elements into guard expressions. Notice that the common elements are `group=general_staff` and `deny`. The following is the same policy, expressed as a layer employing a guard expression:

```
<Proxy> group=general_staff deny
   url.domain=competitor.com/jobs
   url.host=bad_host
   condition=whatever
   ; etc.
   allow
```

Note that the explicit `allow` overrides the `deny` specified in the layer guard. This is an instance of a rule specific property setting overriding the default property settings specified in a guard expression.

Timing

The "late guards early" timing errors that can occur within a rule can arise across rules in a layer. When a trigger cannot yet be evaluated, policy also has to postpone evaluating all following rules in that layer (since if the trigger turns out to be true and the rule matches, then evaluation stops for that layer. If the trigger turns out to be false and the rule misses, then evaluation continues for the rest of the rules in that layer, looking for the first match). Thus a rule inherits the earliest evaluation point timing of the latest rule above it in the layer.

For example, as noted earlier, the following rule would result in a timing conflict error:

```
group=xyz authenticate(MyRealm)

Error: Late condition guards early action: 'authenticate(MyRealm)'
```

The following layer would result in a similar error:

```
<Proxy>
   group=xyz deny
   authenticate(MyRealm)

Error: Late condition 'group=xyz' guards early action: 'authenticate(MyRealm)'
```

This also extends to guard expressions, as the guard condition must be evaluated before any rules in the layer. For example:

```
<Proxy> group=xyz deny
   authenticate(MyRealm)

Error: Late condition 'group=xyz' guards early action: 'authenticate(MyRealm)'
```

Understanding Sections

The rules in layers can optionally be organized in one or more sections, which is a way of grouping rules together. A *section* consists of a section header followed by a list of rules.

Four sections types are supported in a standard CPL file:

- `[Rule]`
- `[url]`
- `[url.domain]`
- `[server_url.domain]`

However, if a CacheOS 4.*x* filter file is used in place of a policy file and running in backward-compatibility mode, the `[Domain-suffix]`, `[Prefix]`, and `[Regular-Expression]` sections are also available. These deprecated sections are described in Appendix E: "Filter File Syntax".

Three of the section types, `[url]`, `[url.domain]` and `[server_url.domain]`, provide optimization for URL tests. The names for these sections correspond to the CPL URL triggers used as the first test for each rule in the section, that is `url=`, `url.domain=` and `server_url.domain=`. The `[url.regex]` section provides factoring and organization benefits, but does not provide any performance advantage over using a `[Rule]` section and explicit `url.regex=` tests.

To give an example, the following policy layer is created:

```
<Proxy>
   url.domain=abc.com/sports deny
```

```
url.domain=nbc.com/athletics deny
; etc, suppose it's a substantial list
url.regex="sports|athletics" access_server(no)
url.regex="\.mail\." deny
; etc
url=www.bluecoat.com/internal group=!bluecoat_employees deny
url=www.bluecoat.com/proteus group=!bluecoat_development deny
; etc
```

This can be recast into three sections:

```
<Proxy>
   [url.domain]
      abc.com/sports deny
      nbc.com/athletics deny
      ; etc.
   [Rule]
      url.regex="sports|athletics" access_server(no)
      url.regex="\.mail\." deny
   [url]
      www.bluecoat.com/internal group=!bluecoat_employees deny
      www.bluecoat.com/proteus group=!bluecoat_development deny
```

Notice that the first thing on each line is not a labelled CPL trigger, but is the argument for the trigger assumed by the section type. Also, after the first thing on the line, the rest of the line is the familiar format.

The performance advantage of using the `[url]`, `[url.domain]`, or `[server_url.domain]` sections is measurable when the number of URLs being tested reaches roughly 100. Certainly for lists of several hundred or thousands of URLs, the performance advantage is significant.

When no explicit section is specified, all rules in a layer are assumed to be in a `[Rule]` section. That is, the first example is equivalent to:

```
<Proxy>
   [Rule]
      url.domain=abc.com/sports deny
      url.domain=nbc.com/athletics deny
      ; etc, suppose it's a substantial list
      url.regex="sports|athletics" access_server(no)
      url.regex="\.mail\." deny
      ; etc
      url=www.bluecoat.com/internal group=!bluecoat_employees deny
      url=www.bluecoat.com/proteus group=!bluecoat_development deny
      ; etc
```

[Rule]

The `[Rule]` section type is used to logically organize policy rules into a section, optionally applying a guard to the contained rules. The `[Rule]` section was so named because it can accept all rules in a policy. If no section is specified, all rules in a layer are assumed to be in a `[Rule]` section.

- Use `[Rule]` sections to clarify the structure of large layers. When a layer contains many rules, and many of the rules have one or more conditions in common, you may find it useful to define `[Rule]` sections.

- Rules in `[Rule]` sections are evaluated sequentially, top to bottom. The time taken is proportional to the number of rules in the section.
- `[Rule]` sections can be used in any layer.

[url]

The `[url]` section type is used to group a number of rules that test the URL. The `[url]` section restricts the syntax of rules in the section. The first token on the rule line is expected to be a pattern appropriate to a `url=` trigger. The trigger name is not included. The `[url]` section replaces the `[Prefix]` section used in previous versions of CPL)

- Rules in `[url]` sections are evaluated through hash table techniques, with the result that the time taken is not dependent on the number of rules in the section.
- `[url]` sections are not allowed in `<Admin>` or `<Forward>` layers.

[url.domain]

The `[url.domain]` section is used to group a number of rules that test the URL domain. The `[url.domain]` section restricts the syntax of rules in the section. The first token on the rule line is expected to be a pattern appropriate to a `url.domain=` trigger. The trigger name is not included. (The `[url.domain]` section replaces the `[domain-suffix]` section used in previous versions of CPL.)

- Rules in `[url.domain]` sections are evaluated through hash table techniques, with the result that the time taken is not dependent on the number of rules in the section.
- `[url.domain]` sections are not allowed in `<Admin>` or `<Forward>` layers.

[url.regex]

The `[url.regex]` section is used to test the URL. The `[url.regex]` section restricts the syntax of rules in the section. The first token on the rule line is expected to be a pattern appropriate to a `url.regex=` trigger. The trigger name is not included . The `[url.regex]` section replaces the `[Regex]` section used in previous versions of CPL.

- Rules in `[url.regex]` sections are evaluated sequentially, top to bottom. The time taken is proportional to the number of rules in the section.
- `[url.regex]` sections are not allowed in `<Admin>` or `<Forward>` layers.

[server_url.domain]

The `[server_url.domain]` section is used to test the domain of the URL used to fetch content from the origin server. The `[server_url.domain]` section restricts the syntax and rules in the section. The first token on the rule line is expected to be a pattern appropriate to a `server_url.domain=` trigger. The trigger name is not included.

`[server_url.domain]` sections contain policy rules very similar to `[url.domain]` sections except that these policy rules test the `server_url`, which reflects any rewrites to the request URL.

- Rules in `[server_url.domain]` sections are evaluated through hash table techniques, with the result that the time taken is not dependent on the number of rules in the section.

- `[server_url.domain]` sections are allowed only in `<Exception>` or `<Forward>` layers.

Section Guards

Just as you can with layers, you can improve policy clarity and maintainability by grouping rules into sections and converting the common conditions and properties into *guard expressions* that follow the section header. A guard expression allows you to take a condition that applies to all the rules and put the common condition next to the section header, as in `[Rule] group=sales`.

Guards are essentially a way of factoring out common sets of triggers and properties, to avoid having to repeat them each time.

Defining Policies

This section includes some guidelines for defining policies using CPL.

- Write an explicit layer header (`<Proxy>`, `<Cache>`, `<Admin>`, `<Forward>`, or `<Exception>`) before writing any rules or sections. The only constructs that should occur before the first layer header are the condition-related definitions and comments.
- Do not begin a policy file with a section, such as `[Rule]`. Ensure all sections occur within layers.
- Do not use `[Rule]` sections unnecessarily.
- Avoid empty or badly formed policy. While some CPL may look well-formed, make sure it actually *does* something.

While the following example appears like proper CPL, it actually has no effect. It has a layer header and a `[Rule]` section header, but no rule lines. As no rules exist, no policy exists either:

```
<Admin> group=Administrators
    [Rule] allow
```

Correct policy that allows access for the group “administrators” would be:

```
<Admin>
    group=Administrators allow
```

In the following example, the layer is deceptive because only the first rule can ever be executed:

```
<Proxy>
   authenticate(MyRealm) ; this rule is unconditional
   ;all following rules are unreachable
   allow group=administrator
   allow group=clerk time=0900..1700
   deny
```

At most, one rule is executed in any given layer. The first one that meets the conditions is acted upon; all other rules in the layer are ignored. To execute more than one rule, use more than one layer. To correctly define the above policy, two layers are required:

```
<Proxy>
   authenticate(MyRealm)
<Proxy>
   allow group=administrator
   allow group=clerk time=0900..1700
   deny
```

- Do not mix the CacheOS 4.*x* filter-file syntax with CPL syntax.

  Although the Content Policy Language is backward-compatible with the filter-file syntax, avoid using the older syntax with the new. For example, as the filter-file syntax uses a different order of evaluation, mixing the old and new syntax can cause problems. Blue Coat strongly recommends not mixing the two syntaxes.

Blacklists and Whitelists

For administrative policy, the intention is to be cautious and conservative, emphasizing security over ease of use. The assumption is that everything not specifically mentioned is denied. This approach, referred to as the *whitelist* approach, is common in corporations concerned with security or legal issues above access. Organizations that want to extend this concept to general Internet access select a default proxy policy of deny as well.

In the second approach, the idea is that by default everything is allowed. Some requests might be denied, but really that is the exception. This is known as *blacklist* policy because it requires specific mention of anything that should be denied (blacklisted). Blacklist policy is used by organizations where access is more important than security or legal responsibilities.

This second approach is used for cache transactions, but can also be common default proxy policy for organizations such as internet service providers.

Blacklists and whitelists are tactical approaches and are not mutually exclusive. The best overall policy strategy is often to combine the two approaches. For example, starting from a default policy of deny, one can use a whitelist approach to explicitly *filter-in* requests that ought to be served in general (such as all requests originating from internal subnets, while leaving external requests subject to the default `DENY`). Further policy layers can then apply more specific restrictions in a blacklist mode to *filter-out* unwanted requests (such as those that fail to conform to content filtering policies).

Whitelisting and blacklisting can also be used not simply to allow or deny service, but also to subject certain requests to further processing. For example, not every file type presents an equal risk of virus infection or rogue executable content. One might choose to submit only certain file types (presumably those known to harbor executable content) to a virus scanner (blacklist), or virus-scan all files except for a whitelist of types (such as image files) that may be considered safe.

General Rules and Exceptions to a General Rule

When writing policy many organizations use general rules, and then define several exceptions to the rule. Sometimes, you might find exceptions to the exceptions. Exceptions to the general rule can be expressed either:

- Through rule order within a layer
- Through layer order within the policy.

Using Rule Order to Define Exceptions

When the policy rules within a layer are evaluated, remember that evaluation is from the top down, but the first rule that matches will end further evaluation of that layer. Therefore, the most specific conditions, or exceptions, should be defined first. Within a layer, use the sequence of most-specific to most-general policy.

The following example is an exception defined within a layer. A company wants access to payroll information limited to Human Resources staff only. The administrator uses membership in the `HR_staff` group to define the exception for HR staff, followed by the general policy:

```
<Proxy>
   ; Blue Coat uses groups to identify HR staff, so authentication is required
   authenticate(MyRealm)

define condition payroll_location
   url=hr.my_company.com/payroll/
end

<Proxy> condition=payroll_location
   allow group=HR_staff ; exception
   deny ; general rule
```

This approach requires that the policy be in one layer, and because layer definitions cannot be split across policy files, the rule and the exceptions must appear in the same file. That may not work in cases where the rules and the exceptions are maintained by different groups.

However, this is the preferred technique, as it maintains all policy related to the payroll files in one place. This approach can be used in either blacklist or whitelist models (see "Blacklists and Whitelists" on page 45) and can be written so that no security holes are opened in error. The example above is a whitelist model, with everything not explicitly mentioned left to the default rule of deny.

Using Layer Ordering to Define Exceptions

Since later layers override earlier layers, general rules can be written in one layer, with exceptions written in following layers, put specific exceptions later in the file.

The Human Resources example could be rewritten as:

```
<Proxy>
   ; Blue Coat uses groups to identify HR staff, so authentication is required
   authenticate(MyRealm)

define condition payroll_location
   url=hr.my_company.com/payroll/
end

<Proxy>
   condition=payroll_location deny ; general rule

<Proxy>
   condition=payroll_location allow group=HR_staff ; exception
```

Notice that in this approach, some repetition is required for the common condition between the layers. In this example, the `condition=payroll_location` must be repeated. It is very important to keep the exception from inadvertently allowing other restrictions to be undone by the use of `allow`.

As the layer definitions are independent, they can be installed in separate files, possibly with different authors. Definitions, such as the payroll location condition, can be located in one file and referenced in another. When linking rules to definitions in other files, file order is not important, but the order of installation is. Definitions must be installed before policy that references them will compile. Keeping definitions used across files in only one of the files, rather than spreading them out, will eliminate the possibility of having changes rejected because of interlocking reference problems. Note that when using this approach, exceptions must follow the general rule, and you must be aware of the policy file

evaluation order as currently configured. Changes to the policy file evaluation order must be managed with great care.

Remember that properties maintain any setting unless overridden later in the file, so you could implement general policy in early layers by setting a wide number of properties, and then use a later layer to override selected properties.

Avoid Conflicting Actions

Although policy rules within a policy file can set the action property repeatedly, turning individual actions on and off for the transaction being processed, the specific actions can conflict.

- If an action-definition block contains two conflicting actions, a compile-time error results. This conflict would happen if, for example, the action definition consisted of two `response.icap_service( )` actions.
- If two different action definitions are executed and they contain conflicting actions, it is a run-time error; a policy error is logged to the event log, and one action is arbitrarily chosen to execute.

The following describes the potential for conflict between various actions:

- Two header modification actions will conflict if they modify the same header. Header modification actions include the `append( )`, `delete( )`, `delete_matching( )`, `rewrite(`*`header`*`,...)`, and `set(`*`header`*`,...)` actions.
- Two instant message text modification actions will conflict. Instant message text modification actions include the `append(im.message.text,...)` and `set(im.message.text,...)` actions.
- Two transform actions of the same type conflict.
- Two `rewrite( )` actions conflict.
- Two `response.icap_service( )` actions conflict.

Making Policy Definitive

You can make policy definitive two ways. The first is to put that policy into the file; that is, last in the evaluation order. (Remember that the forward file is always the last policy file.) For example, suppose that service was limited to the corporate users identifiable by subnet. Placing a rule such as:

```
<Proxy>
   client.address=!my_subnet deny
```

at the end of the Forward file ensures that no other policy overrides this restriction through accidental use of allow. Although not usually used for this purpose, the fact that the forward file is always last, and the order of the other three files is configurable, makes this the appropriate location for definitive policy in some organizations.

An alternate method has been provided for definitive denial. While a `deny` or an `exception()` property can be overridden by a subsequent allow in later rules, CPL provides `force_deny` and `force_exception()`. CPL does not offer `force_allow`, so while the error returned to the user may be reset by subsequent `force_deny` or `force_exception()` gestures, the ultimate effect is that the request is denied. Thus these properties provide definitive denial regardless of where they appear in policy.

Best Practices

- Express separate decisions in separate layers.

  As policy grows and becomes more complex, maintenance becomes a significant issue. Maintenance will be easier if the logic for each aspect of policy is separate and distinct.

  Try to make policy decisions as independent as possible, and express each policy in one layer or two adjacent layers.

- Be consistent with the model.

  Set the default proxy policy according to your policy model and then use blacklist or whitelist approaches as appropriate.

  The recommended approach is to begin with a default proxy policy of deny in configuration. Allow requests in early layers and deny requests in later layers. Ensure that all layers that allow requests precede any layers that deny requests. The following is a simple illustration of this model:

```
define subnet corporate_subnet
   10.10.12.0/24
end

; First, explicitly allow access to our users
<proxy>
   ALLOW client.address=corporate_subnet

; Next, impose any authentication requirements
<proxy>
   authenticate(corp_realm)  ; all access must be authenticated

; And now begin to filter-out unwanted requests
<proxy>
   DENY url.domain=forbidden.com
   DENY category=(Gambling, Hacking, Chat)

   ; more layers…
```

- Expose only what is necessary.

  Often, it may be useful to keep the rule logic and the condition definitions separate so that the rules can be maintained by one group, but the definitions by another. The rules may contain exception details or other logic that should not be modified; however, the conditions, such as affected groups or content, may change frequently. With careful separation of the rules and the conditions, the rules can be expressed in the local policy file, and users unfamiliar with CPL can update the condition definitions through the VPM.

  When using this technique, installation order is important. Condition definitions must be available before policy referencing those conditions will compile, so the conditions you want to expose for general use must be defined in the VPM before they are referenced in the Local policy file.

Chapter 3: *Condition Reference*

A *condition* is an expression that yields true or false when evaluated. Conditions can appear in:

- Policy rules.
- Section and layer headers, as guards; for example,

  ```
  [Rule] group=(“bankabc\hr” || “cn=humanresources,ou=groups,o=westernnational”)
  ```

- `define condition`, `define domain condition`, and `define prefix condition` definition blocks.

Condition Syntax

A condition has the following form:

```
trigger=pattern-expression
```

A *trigger* is the name of a condition variable. It can be simple, such as `url`, or it can contain sub-object specifiers and modifiers, as in `url.path.case_sensitive` or `request.header.Cookie`. A trigger cannot contain white space.

A *pattern expression* can be either:

- A simple pattern, which is matched against the trigger value.
- A Boolean combination of simple patterns, or a parenthesized, comma-separated list of simple patterns.

A pattern expression can be any of the following:

- String: A string argument must be quoted if it contains whitespace or other special characters. An example condition expression is `category=”self help”`.
- Single argument: Conditions such as `live=` take only a single argument, in this case, `yes` or `no`.
- Boolean expressions: Conditions such as `server_url.scheme=` can list one or more arguments together with Boolean operators; for example, `server_url.scheme=!http`.
- Integer or range of integers: Numeric conditions can use Boolean expressions and double periods (`..`), meaning an inclusive numeric range. Numeric ranges *cannot* use whitespace. The `minute=` condition is used to show examples of ranges:
  - `minute=10..40`–From 10 minutes to 40 minutes after the hour.
  - `minute=10..`–From 10 minutes after the hour to the end of the hour.
  - `minute=..40`–From the beginning of the hour to 40 minutes after the hour.
  - `minute=40..10`—From 40 minutes after the hour, to 10 minutes after the next hour.
- Regular expressions: Some header-related conditions and two URL-related conditions take regular expressions. For more information about writing regular expressions, refer to Appendix E: “Using Regular Expressions,” in the Blue Coat Proxy*SG Configuration and Management Guide*.

The following is Backus-Naur Form (BNF) grammar:

- `condition ::= trigger "=" expression`
- `trigger ::= identifier | identifier "." word`
- `expression ::= term | list`
- `list ::= "(" ((pattern ",")* pattern)? ")"`
- `disjunction ::= conjunction | disjunction "||" conjunction`
- `conjunction ::= term | conjunction "&&" term`
- `term ::= pattern | "(" disjunction ")" | "!" term`
- `pattern ::= word | 'string' | "string"`
- `word ::= sequence of characters not including whitespace, & | ( ) < > [ ] ; ! = " '`
- `string ::= sequence of characters that may including whitespace, & | ( ) < > [ ] ; ! =. The characters " and ' may be enclosed within a string delimited by the alternate delimiter.`

Pattern Types

Different triggers support different pattern syntaxes.

A pattern for a boolean trigger has one of the following forms:

```
boolean ::= yes | no | true | false | on | off
```

The pattern for a numeric trigger can be either an integer or a range of integers. Numeric patterns *cannot* contain white space.

`trigger=I`

Test if trigger `== I`.

`trigger=I..J`

Test if trigger `>= I` and trigger `<= J` (where `I <= J`). For example, `time=0900..1700` tests if the time is between 9:00 and 17:00 inclusive.

`trigger=J..I`

Test if trigger `>= J` or trigger `<= I` (where `J > I`). For example, `minute=45..15` tests if the minute of the hour is between 45 and 15 inclusive.

`trigger=I..`

Test if trigger `>= I`. For example, `bitrate=56k..` tests if the bitrate is greater than or equal to 56000.

`trigger=..J`

Test if trigger `<= J`. For example, `bitrate=..56k` tests if the bitrate is less than or equal to 56000.

Some triggers have IP address patterns. This can be either a literal IP address, such as 1.2.3.4, or an IP subnet pattern, such as 1.2.0.0/16, or a name defined by a ***`define subnet`*** statement.

Some triggers have regex patterns. This is a Perl 5 regular expression that matches a substring of the trigger value; it is not an anchored match unless an anchor is specified as part of the pattern.

Unavailable Triggers

Some triggers can be unavailable in some transactions. If a trigger is unavailable, then any condition containing that trigger is false, regardless of the pattern expression. For example, if the current transaction is not authenticated (that is, the authenticate property was set to `no`), then the `user` trigger is unavailable. This means that `user=kevin` and `user=!kevin` are both false.

A condition can be false either because the pattern does not match the trigger value, or because the trigger is unavailable. Policy rule-tracing distinguishes these two cases, using `miss` for the former and `N/A` for the latter.

Layer Type Restrictions

Each trigger is restricted as to the types of layers in which it can be used. A direct use of a trigger in a forbidden layer results in a compile-time error. Indirect use of a trigger in a forbidden layer (by way of `condition=` and a condition definition) also results in a compile time error.

Global Restrictions

To allow suppression of DNS and RDNS lookups from policy, the following restrictions are supported. These restrictions have the effect of assuming a `no_lookup` modifier for appropriate `url=` and `server_url` tests. The restrictions also apply to lookups performed by on-box content category lookups. For more information on DNS and RDNS restrictions, see Chapter 6: "Definition Reference".

`restrict dns`
`domain_list`
`end`
	Applies to all layers.	Applies to all transactions.	If the domain specified in a URL matches any of the domain patterns specified in `domain_list`, no DNS lookup is performed for any `server_url=`, `server_url.address=`, `server_url.domain=`, or `server_url.host= test`.
			If a lookup is required to evaluate the trigger, the trigger evaluates to `false`.

`restrict rdns`
`subnet_list`
`end`
	Applies to all layers.	Applies to all transactions.	If the requested URL specifies the host in IP form, no RDNS lookup is performed to match any `server_url=`, `server_url.domain=`, or `server_url.host= trigger`.
			If a lookup is required to evaluate the trigger, the trigger evaluates to `false`.

Condition Reference

The remainder of this chapter lists the conditions and their accepted values. It also provides tips as to where each condition can be used and examples of how to use them.

acl=

Deprecated syntax. See "client.address=" on page 60 for more information.

admin.access=

Tests the administrative access requested by the current transaction.

It evaluates to null if the transaction is not an administrative transaction, which may occur if the test is included in an `<Exception>` layer.

Replaces: `method=`

Syntax

```
admin.access=READ|WRITE
```

Layer and Transaction Notes

- Use in `<Admin>` layers instead of method=
- Applies to administrator transactions.

Examples

This example grants full administrative access to members of the IT_Admin group, allows read-only access to members of the IT group, and denies administrative access to all others.

```
<Admin>
   authenticate(MyRealm)

<Admin>
group=IT_Admin allow
group=IT_support admin.access=READ allow ; can view but not modify
deny
```

See Also

- Conditions: `console_access=`

attribute.*name*=

Tests if the current transaction is authenticated in a RADIUS or LDAP realm, and if the authenticated user has the specified attribute with the specified value. This trigger is unavailable if the current transaction is not authenticated (that is, the `authenticate` property is set to `no`).

If you reference more than one realm in your policy, you may wish to disambiguate attribute tests by combining them with a realm= test. This can reduce the number of extraneous queries to authentication services for attribute information that does not pertain to that realm.

Syntax

```
attribute.name=value
```

where:

- *name* is a RADIUS or LDAP attribute. The *name* attribute's case-sensitivity depends on the type of authentication realm.
- RADIUS realm: The only available attribute is `ServiceType`, which is always case-sensitive.
- LDAP realm: Case-sensitivity depends on the realm definition in configuration.
- *value*: An attribute value.

Layer and Transaction Notes

- Use in `<Admin>` and `<Proxy>` layers.
- Applies to proxy and administrator transactions.
- This condition cannot be combined with the `authenticate()` or `socks.authenticate()` properties.

Examples

```
; This example uses the value of the ContentBlocking attribute associated with a
; user to select which content categories to block. (SmartFilter 3 categories are
; used.)

<proxy>

authenticate(LDAPRealm)

<proxy> exception(content_filter_denied)

attribute.ContentBlocking=Adult category=(Sex, Nudity, Mature, Obscene/Extreme)
attribute.ContentBlocking=Violence category=(Criminal_Skills, Hate_Speech)

...


; This example uses the attribute property to determine permissions associated with
; RADIUS authentication.

define condition ProxyAllowed

   attribute.ServiceType=(2,6,7,8)

end
```

```
<proxy>
authenticate(RADIUSRealm)

; This rule would restrict non-authorized users.
<proxy>
deny condition=!ProxyAllowed

; This rule would serve to override a previous denial and grant access to authorized
; users
<proxy>
allow condition=ProxyAllowed
```

See Also

- Conditions: `authenticated=`, `group=`, `has_attribute.`*`name`*`=`, `http.transparent_authentication=`, `realm=`, `user=`, `user.domain=`
- Properties: `authenticate( )`, `authenticate.force( )`, `check_authorization( )`, `exception( )`, `socks.authenticate( )`, `socks.authenticate.force( )`

authenticated=

True if authentication was requested and the credentials could be verified; otherwise, false.

Syntax

```
authenticated=(yes|no)
```

Layer and Transaction Notes

- Use in <Admin> and <Proxy> layers.
- Applies to proxy and administrator transactions.
- This condition cannot be combined with the authenticate() property.

Examples

```
; In this example, only users authenticated in any domain are granted access to a
; specific site.

<proxy>

client.address=10.10.10.0/24 authenticate(LDAPRealm)
client.address=10.10.11.0/24 authenticate(NTLMRealm)
client.address=10.10.12.0/24 authenticate(LocalRealm)
;anyone else is unauthenticated


; This rule would restrict unauthorized users. Use this when overriding previously
; granted access.

<proxy> server_url.domain=xyz.com

deny authenticated=no


; This rule would grant access and would be used to override a previous denial.
; It assumes a deny in a previous layer.

<proxy> server_url.domain=xyz.com

allow authenticated=yes
```

See Also

- Conditions: attribute.*name*=, group=, has_attribute.*name*=, http.transparent_authentication=, realm=, user=, user.domain=
- Properties: authenticate( ), authenticate.force( ), check_authorization(), socks.authenticate( ), socks.authenticate.force( )

bitrate=

Tests if a streaming transaction requests bandwidth within the specified range or an exact match. When providing a range, either value can be left empty, implying either no lower or no upper limit on the test. Bitrate can change dynamically during a transaction, so this policy is re-evaluated for each change. Note that the numeric pattern used to test the `bitrate=` condition can contain no whitespace. This trigger is only available if the current transaction is a Real Media or Windows Media transaction.

Syntax

```
bitrate={ [lower]..[upper]|exact_rate }
```

where:

- *`lower`*—Lower end of bandwidth range. Specify using an integer, in bits, kilobits (1000x), or megabits (1,000,000x), as follows: `integer | integer`*`k`* `| integer`*`m`*. If left blank, there is no lower limit on the test.
- *`upper`*—Upper end of bandwidth range. Specify using an integer, in bits, kilobits, or megabits, as follows: `integer | integer`*`k`* `| integer`*`m`*. If left blank, there is no upper limit on the test.
- *`exact_rate`*—Exact bandwidth to test. Specify using an integer, in bits, kilobits, or megabits, as follows: `integer | integer`*`k`* `| integer`*`m`*.

Note: To test an inverted range, the following shorthand expression is available. Instead of writing `bitrate=(..28.8k|56k..)` to indicate bit rates from 0 to 28.8k and from 56k up, the policy language recognizes `bitrate=56k..28.8k` as equivalent.

Layer and Transaction Notes

- Use in `<Cache>` and `<Proxy>` layers.
- Applies to streaming transactions.
- This condition can be used with the `max_bitrate( )` property.

Examples

```
; Deny service for bit rates above 56k.

deny bitrate=!0..56k

; This example allows members of the Sales group access to streams up to 2 megabits.
; All others are limited to 56K bit streams.

<proxy>
    authenticate(NTLMRealm)

<proxy>
    ; deny sales access to streams over 2M bits
    deny group=sales bitrate=!0..2m

    ; deny non-sales access to streams over 56K bits
    deny group=!sales bitrate=!0..56k..


; In this form of the rule, we assume that the users are by default denied, and we
; are overriding this to grant access to authorized users.
```

```
<Proxy> ; Use this layer to override a deny in a previous layer
    ; Grant everybody access to streams up to 56K, sales group up to 2M
    allow bitrate=..56K
    allow group=sales bitrate=..2M
```

See Also

- Conditions: `live=`, `streaming.client=`, `streaming.content=`
- Properties: `access_server( )`, `max_bitrate( )`, `streaming.transport( )`

category=

Tests the content categories of the requested URL as assigned by policy definitions or an installed content filter database.

A URL that is not categorized is assigned the category `none`.

If a content filter provider is selected in configuration, but an error occurs in determining the category, the URL is assigned the category `unavailable` (in addition to any categories assigned directly by policy). This can be the result of either a missing database or license expiry. An additional category of `unlicensed` is assigned in the latter case.

A URL may have been assigned a list of categories. The `category=` trigger is true if it matches any of the categories assigned to the URL.

You cannot use `category=` to test the category assigned by off-box content filtering services. These services have their own policy that must be managed separately.

Notes:

- If `category=unlicensed` is true, `category=unavailable` is true.
- `category=unavailable` replaces the deprecated `category.unavailable=yes` syntax.
- `category=(`*`category_list`*`) exception(content_filter_denied)` replaces the deprecated `block_category(`*`category_list`*`)` syntax.

Syntax

```
category={ none|unlicensed|unavailable|category_name1, category_name2, ...}
```

where *`category_name1, category_name2, ...`* represent category names defined by policy or the selected content filter provider. The list of currently valid category names is available both through the Management Console and CLI.

Layer and Transaction Notes

- Use in `<Cache>`, `<Proxy>`, and `<Exception>` layers.
- This condition can be combined with the `authenticate( )` property, except when a Microsoft Media Streaming (MMS) over HTTP transaction is being evaluated.
- Applies to proxy transactions.

Examples

```
; This example denies requests for games or sports related content.
<Proxy>
    ; Tests true if the request is in one of these categories.
    category=(Sports, Games) exception(content_filter_denied)
    category=unavailable exception(content_filter_unavailable); Fail closed
```

See Also

- Properties: `exception( )`, `request.filter_service( )`

client.address=

Tests the IP address of the client. The expression can include an IP address or subnet or the label of a subnet definition block.

Important: If a user is explicitly proxied to the Proxy*SG*, `<Proxy>` layer policy applies even if the URL destination is an administrative URL for the Proxy*SG* itself, and should therefore also be covered under `<Admin>` layer policy. However, when the `client.address=` trigger is used in an `<Admin>` layer, clients explicitly proxied to the Proxy*SG* appear to have their client IP address set to the IP address of the Proxy*SG*.

Replaces: `client_address=`, `acl=`

Syntax

```
client.address=ip_address|subnet_label
```

where:

- *`ip_address`*—Client IP address or subnet specification; for example, `10.25.198.0/24`.
- *`subnet_label`*—Label of a subnet definition block that binds a number of IP addresses or subnets.

Layer and Transaction Notes

- Can be used in all layers.
- Unavailable if the transaction is not associated with a client.

Examples

```
; Blacklisted workstation.
client.address=10.25.198.0 deny


; This example uses the client address to select the authentication realm for
; administration of the ProxySG.
<admin>
client.address=10.25.198.0/24 authenticate(LDAPRealm)
client.address=10.25.199.0/24 authenticate(NTLMRealm)
authenticate(LocalRealm) ; Everyone else
```

See Also

- Conditions: `client.protocol=`, `proxy.address=`, `proxy.card=`, `proxy.port=`
- Definitions: `define subnet`

client.protocol=

Tests true if the client transport protocol matches the specification.

Replaces: `client_protocol=`

syntax

```
client.protocol=http|https|ftp|tcp|socks|mms|rtsp|icp|aol-im|msn-im|yahoo-im
```

Note that `tcp` specifies a tunneled transaction.

Layer and Transaction Notes

- Use in `<Exception>`, `<Forward>`, and `<Proxy>` layers.
- Applies to proxy transactions.
- Tests false if the transaction is not associated with a client.

See Also

- Conditions: `client.address=`, `proxy.address=`, `proxy.card=`, `proxy.port=`

condition=

Tests if the specified defined condition is true.

Syntax

```
condition=condition_label
```

where *condition_label* is the label of a custom condition as defined in a `define condition`, `define url.domain condition`, or `define url condition` definition block.

Layer and Transaction Notes

- Use in all layers.
- The defined conditions that are referenced may have usage restrictions, as they must be evaluated in the layer from which they are referenced.

Examples

```
; Deny access to client 1.2.3.4 for any http request through proxy port 8080.
define condition qa
    client.address=1.2.3.4 proxy.port=8080
end condition qa
<proxy>
condition=qa client.protocol=http deny


; Restrict access to internal sites to specific groups,
; using nested conditions.
define condition restricted_sites
    url.domain=internal.my_co.com
end condition restricted_sites
define condition has_full_access
    group=admin,execs,managers
end condition
define condition forbidden
    condition=restricted_sites condition=!has_full_acesss
end
<proxy>
    authenticate(My_realm)
<proxy>
    condition=forbidden deny


; Example of a define url condition.
define url condition test
```

```
   http://www.x.com time=0800..1000
   http://www.y.com month=1
   http://www.z.com hour=9..10
end
<proxy>
    condition=test deny


; Example of a define domain-suffix (or domain) condition
define url.domain condition test
   com ; Matches all domains ending in .com
end
<proxy>
   condition=test deny
```

See Also

- Definitions: `define condition`, `define url.domain condition`, `define url condition`
- Properties: `action.`*`action_label`*`( )`

console_access=

Tests if the current request is destined for the <Admin> layer. This test can be used to distinguish access to the management console by admininstrators who are explicitly proxied to the ProxySG being admininstered. The test can be used to guard transforms that should not apply to the Management Console. This cannot be used to test Telnet sessions, as they do not go through a <Proxy> layer.

Syntax

```
console_access=yes|no
```

Layer and Transaction Notes

- Use in <Exception>, <Proxy>, and <Cache> layers.
- Applies to HTTP transactions.

See Also

- Conditions: admin.access=

content_admin=

The `content_admin=` condition has been deprecated. For more information, see "content_management" on page 66.

content_management

Tests if the current request is a content management transaction.

Replaces: `content_admin=yes|no`

Syntax

```
content_management=yes|no
```

Layer and Transaction Notes

- Use in `<Cache>` and `<Forward>` layers.
- Applies to all transactions.

See Also

- Conditions: `category=`, `ftp.method=`, `http.method=`, `http.x_method=`, `method=`, `server_url=`
- Properties: `http.request.version( )`, `http.response.version( )`

date[.utc]=

Tests true if the current time is within the `startdate..enddate` range, inclusive. The comparison is made against local time unless the `.utc` qualifier is specified.

syntax

```
date[.utc]=YYYYMMDD..YYYYMMDD
date[.utc]=MMDD..MMDD
```

Layer and Transaction Notes

- Using time-related conditions to control caching behavior in a `<Cache>` layer may cause thrashing of the cached objects.

See Also

- Conditions: `day=`, `hour=`, `minute=`, `month=`, `time=`, `weekday=`, `year=`

day=

Tests if the day of the month is in the specified range or an exact match. The ProxySG appliance's configured date and time zone are used to determine the current day of the month. To specify the UTC time zone, use the form `day.utc=`. Note that the numeric pattern used to test the `day` condition can contain no whitespace.

Syntax

```
day[.utc]={[first_day]..[last_day]|exact_day}
```

where:

- *first_day*—An integer from 1 to 31, indicating the first day of the month that will test true. If left blank, day 1 is assumed.
- *last_day*—An integer from 1 to 31, indicating the last day of the month that will test true. If left blank, day 31 is assumed.
- *exact_day*—An integer from 1 to 31, indicating the day of the month that will test true.

Note: To test against an inverted range, such as days early and late in the month, the following shorthand expression is available. While `day=(..5|25..)` specifies the first 5 days of the month and last few days of the month, the policy language also recognizes `day=25..5` as the same.

Layer and Transaction Notes

- Use in all layers.
- Using time-related conditions to control caching behavior in a `<Cache>` layer may cause thrashing of the cached objects.

Examples

```
; Test for New Year's Day (January 1).

day=1 month=1

; This policy allows access to a special event site only during the days of
; the event.

; This form of the rule restricts access during non-event times.

<Proxy> url=http://www.xyz.com/special_event

; The next line matches, but does nothing if allow is the default
; year=2003 month=7 day=23..25 ; During the event
; deny Any other time

; This form of the rule assumes access is generally denied, and grants access during
; the special event.

<Proxy> url=http://www.xyz.com/special_event

allow year=2003 month=7 day=23..25 ; During the event
```

See Also

- Conditions: `date[.utc]=`, `hour=`, `minute=`, `month=`, `time=`, `weekday=`, `year=`

exception.id=

Tests whether the exception being returned to the client is the specified exception. It can also be used to determine whether the exception being returned is a built-in or user-defined exception.

Built-in exceptions are handled automatically by the Proxy*SG* but special handling can be defined within an `<Exception>` layer. Special handling is most often required for user-defined exceptions.

syntax

```
exception.id=exception_id
```

where *exception_id* is either the name of a built-in exception of the form:

```
exception_id
```

or the name of a user defined exception in the form:

```
user_defined.exception_id
```

In addition to testing the identity of exceptions set by the `exception( )` property, `exception.id=` can also test for exceptions returned by other CPL gestures, such as `policy_denied`, returned by the `deny( )` property and `policy_redirect` returned by the `redirect( )` action.

Layer and Transaction Notes

- Use in `<Exception>` layers.
- Applies to proxy transactions.

Examples

This example illustrates how some commonly generated exceptions are caught. Appropriate subnet and action and category definitions are assumed.

```
<Proxy> url.domain=partner.my_co.com/
   action.partner_redirect(yes) ; action contains redirect( )
<Proxy> url.domain=internal.my_co.com/
   force_deny client.address!=mysubnet
   authenticate(my_realm)
<Proxy> deny.unauthorized
   url.domain=internal.my_co.com/hr group=!hr;
   ; and other group/user restrictions ...
<Proxy> category=blocked_sites
   exception(user_defined.restricted_content )
   ; could probably have used built in content_filter_denied
; Custom handling for some built-in exceptions
;
<Exception>
   ; thrown by authenticate( ) if there is a realm configuration error
   exception.id=configuration_error action.config_err_alerts(yes)
   ; thrown by deny.unauthorized
   exception.id=authorization_failed action.log_permission_failure(yes)
```

```
   ; thrown by deny or force_deny
   exception.id=policy_denied action.log_interloper(yes)

<Exception> exception.id=user_defined.restricted_content

   ; any policy required for this user defined exception
   ...
```

See Also

- Properties: `deny( )`, `deny.unauthorized( )`, `exception( )`
- Actions: `authenticate( )`, `authenticate.force( )`, `redirect( )`

ftp.method=

Tests FTP request methods against any of a well-known set of FTP methods. A CPL parse error is given if an unrecognized method is specified.

- `ftp.method=` evaluates to true if the request method matches any of the methods specified.
- `ftp.method=` evaluates to NULL if the request is not an FTP protocol request.

Syntax

```
ftp.method=ABOR|ACCT|ALLO|APPE|CDUP|CWD|DELE|HELP|LIST|MDTM|MKD|MODE|NLST|NOOP|P
ASS|PASV|PORT|PWD|REST|RETR|RMD|RNFR|RNTO|SITE|SIZE|SMNT
|STOR|STOU|STRU|SYST|TYPE|USER|XCUP|XCWD|XMKD|XPWD|XRMD|OPEN
```

where:

- `ftp.method=` evaluates to true if the request method matches any of the methods specified.
- It evaluates to NULL if the request is not an FTP protocol request.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, and `<Exception>` layers.
- Applies to FTP transactions.

See Also

- Conditions: `category=`, `content_management=`, `http.method=`, `http.x_method=`, `im.method=`, `method=`, `server_url=`, `socks.method=`

group=

Tests if the client is authenticated, and the client belongs to the specified group. If both of these conditions are met, the result is true. In addition, the `realm=` condition can be used to test whether the user is authenticated in the specified realm. This trigger is unavailable if the current transaction is not authenticated; that is, the `authenticate( )` property is set to `no`.

If you reference more than one realm in your policy, consider disambiguating group tests by combining them with a `realm=` test. This reduces the number of extraneous queries to authentication services for group information that does not pertain to that realm.

Syntax

```
group=group_name
```

where:

- *group_name*—Name of a group in the default realm. The required form, and the *name* attribute's case-sensitivity, depends on the type of realm.
  - NTLM realm: Group names are of the form `Domain\groupname`, where `Domain` may be optional, depending on whether or not the CAASNT is installed on the NT domain controller for the domain. Names are case-insensitive.
  - Local Password realm: Group names are up to 32 characters long, and underscores (_) and alphanumerics are allowed. Names are case-sensitive.
  - RADIUS realm: RADIUS does not support groups. Instead, *groups* in RADIUS environments are defined by assigning users a `ServiceType` attribute.
  - LDAP realm: Group definitions depend on the type of LDAP directory and LDAP schema. Generally, LDAP distinguished names are used in the following form: `cn=proxyusers`, `ou=groups`, `o=companyname`. Case-sensitivity depends on the realm definition configuration.
  - Certificate realm: Certificate realms provide authentication, but do not themselves provide authorization; instead they delegate group membership decisions to their configured authorization realm, which is either a Local Password realm or an LDAP realm. Group definitions should conform to the appropriate standards for the delegated authorization realm. Although the group used in policy is then a group from the delegated realm, to achieve performance benefits, the `group=` test should be preceded with a realm test for the certificate realm, not the delegated authorization realm.
  - Sequence realm: A sequence realm is a configured list of subordinate realms to which the user credentials are offered, in the order listed. The user is considered authenticated when the offered credentials are valid in one of the realms in the sequence. Authorization of the user is done with respect to the subordinate realm in which authentication occurred. Group names may be valid names in any of the realms in the sequence, but for the `group=` test to evaluate to true, the group must be valid in the realm in which the user is actually authenticated. If the group is valid in all realms in the sequence, then the `group=` test must be preceded by a `realm=` test of the Sequence realm; otherwise, it should be preceded by a `realm=` test of the appropriate subordinate realm.

Layer and Transaction Notes

- Use in `<Admin>` and `<Proxy>` layers.

- Applies to proxy and administrator transactions.
- This condition cannot be combined with the `authenticate( )`, `proxy_authentication( )`, or `socks.authenticate( )` properties.

Examples

```
; Test if user is authenticated in group all_staff and specified realm.

realm=corp group=all_staff


; This example shows sample group tests for each type of realm. It does
; this by creating a condition in CPL that treats a group of administrators in
; each realm as equivalent, granting them permission to administer the Security
; Appliance. Recall that the <Admin> layer uses a whitelist model by default.

define condition RW_Admin

   realm=LocalRealm group=RWAdmin
   realm=NTLMRealm group=xyz-domain\cache_admin
   realm=LDAPRealm group=”cn=cache_admin, ou=groups, o=xyz”
   ; The RADIUSRealm uses attributes, and this can be expressed as follows:
   realm=RADIUSRealm attribute.ServiceType=8

end condition RW_Admin

<admin>

   client.adress=10.10.1.250/28 authenticate(LocalRealm)
   client.adress=10.10.1.0/24 authenticate(NTLMRealm)
   client.adress=10.10.2.0/24 authenticate(LDAPRealm)
   client.adress=10.10.3.0/24 authenticate(RADIUSRealm)

<admin>

   allow condition=RW_Admin admin.access=(READ||WRITE)
```

See Also

- Conditions: `attribute.name=`, `authenticated=`, `has_attribute.`*`name`*`=`, `http.transparent_authentication=`, `realm=`, `user=`, `user.domain=`
- Properties: `authenticate( )`, `authenticate.force( )`, `check_authorization( )`, `socks.authenticate( )`, `socks.authenticate.force( )`

has_attribute.*name*=

Tests if the current transaction is authenticated in an LDAP realm and if the authenticated user has the specified LDAP attribute. If the attribute specified is not configured in the LDAP schema and `yes` is used in the expression, the condition always yields false. This trigger is unavailable if the current transaction is not authenticated (that is, the `authenticate` property is set to `no`).

If you reference more than one realm in your policy, consider disambiguating `has_attribute` tests by combining them with a `realm=` test. This reduces the number of extraneous queries to authentication services for attribute information that does not pertain to that realm.

Important: This condition is incompatible with Novell eDirectory servers. If the *name* attribute is configured in the LDAP schema, then all users are reported by the eDirectory server to have the attribute, regardless of whether they actually do. This can cause unpredictable results.

Syntax

```
has_attribute.name=yes|no
```

where *name* is an LDAP attribute. Case-sensitivity for the attribute name depends on the realm definition in configuration.

Layer and Transaction Notes

- Use in `<Admin>` and `<Proxy>` layers.
- Applies to proxy and administrate transactions.
- This condition cannot be combined with the `authenticate( )`or `socks.authenticate( )` properties.

Example

```
; The following policy allows users to access the proxy if they have the
; LDAP attribute ProxyUser. The attribute could have any value, even null.
; Generally this kind of policy would be established in the first proxy layer,
; and would set up either the blacklist or whitelist model, as desired.

<proxy>
   authenticate(LDAPRealm)


; Setting up a whitelist model

<proxy>

deny has_attribute.ProxyUser=no


; Setting up a blacklist model

<proxy>
   allow has attribute.ProxyUser=yes

deny
```

See Also

- Conditions: `attribute.`*`name`*`=`, `authenticated=`, `group=`, `http.transparent_authentication=`, `realm=`, `user=`, `user.domain=`
- Properties: `authenticate( )`, `authenticate.force( )`, `check_authorization( )`

has_client=

The `has_client=` condition is used to test whether or not the current transaction has a client. This can be used to guard triggers that depend on client identity in a `<Forward>` layer.

Syntax

```
has_client=yes|no
```

Layer and Transaction Notes

- Use in `<Forward>` layers.
- Applies to all transactions.

See Also

- Conditions: `client.address=`, `client.protocol=`, `proxy.address=`, `proxy.card=`, `proxy.port=`, `streaming.client=`

hour=

Tests if the time of day is in the specified range or an exact match. The current time is determined by the Proxy*SG* appliance's configured clock and time zone by default, although the UTC time zone can be specified by using the form `hour.utc=`. The numeric pattern used to test the `hour=` condition contains no whitespace.

Note: Any range of hours or exact hour includes all the minutes in the final hour. See the "Examples" section.

Syntax

```
hour[.utc]={first_hour]..[last_hour]|exact_hour}
```

where:

- *`first_hour`*—Two digits (*nn*) in 24-hour time format representing the first hour in a range; for example, 09 means 9:00 a.m. If left blank, midnight (`00`) is assumed—exactly 00:00 a.m.
- *`last_hour`*—Two digits (*nn*) in 24-hour time format representing the last full hour in a range; for example, `17` specifies 5:59 p.m. If left blank, `23` is assumed (23:59 p.m.).
- *`exact_time`*—Two digits (*nn*) in 24-hour time format representing an exact, full hour.

Note: To test against an inverted range, such as a range that crosses from one day into the next, the following shorthand expression is available. While `hour=(..06|19..)` specifies midnight to 6:59 a.m. and 7:00 p.m. to midnight, the policy language also recognizes `hour=19..06` as equivalent.

Layer and Transaction Notes

- Use in all layers.
- Using time-related conditions to control caching behavior in a `<Cache>` layer may cause thrashing of the cached objects.
- Applies to all transactions.

Examples

```
; Tests for 3:00 a.m. to 1:59 p.m. UTC.

hour.utc=03..13


; The following example restricts access to external sites during business hours.
; This rule assumes that the user has access that must be restricted.

<proxy>

   ; Internal site always available, no action required
   server_url.domain=xyz.com
   ; Restrict other sites during business hours
   deny weekday=1..5 hour=9..16


; If a previous rule had denied access, then this rule could provide an exception.
```

```
<proxy>
   allow server_url.domain=xyz.com ; internal site always available
      allow weekday=6..7       ; unrestricted weekends
      allow hour=17..8; Inverted range for outside business hours
```

See Also

- Conditions: `date[.utc]=`, `day=`, `minute=`, `month=`, `time=`, `weekday=`, `year=`

http.method=

Tests HTTP request methods against any of a common set of HTTP methods. A CPL parse error is given if an unrecognized method is specified.

Syntax

```
http.method=GET|CONNECT|DELETE|HEAD|POST|PUT|TRACE|OPTIONS|TUNNEL|LINK|UNLINK
|PATCH|PROPFIND|PROPPATCH|MKCOL|COPY|MOVE|LOCK|UNLOCK|MKDIR|INDEX|RMDIR|COPY|
MOVE
```

where:

- `http.method=` evaluates to true if the request method matches any of the methods specified.
- `http.method=` evaluates to NULL if the request is not an HTTP protocol request.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, and `<Exception>` layers.
- Applies to HTTP transactions.

See Also

- Conditions: `admin.access=`, `ftp.method=`, `http.x_method=`, `im.method=`, `method=`, `socks.method=`
- Properties: `http.request.version( )`, `http.response.version( )`

http.request.version=

Tests the version of HTTP used by the client in making the request to the appliance.

syntax

```
http.request.version=0.9|1.0|1.1
```

Layer and Transaction Notes

- Use in <Proxy>, <Cache>, and <Exception> layers.
- Applies to HTTP transactions.

See Also

- Conditions: http.response.code=, http.response.version=
- Properties: http.request.version( ), http.response.version( )

http.response.code=

Tests true if the current transaction is an HTTP transaction and the response code received from the origin server is as specified.

Replaces: `http.response_code`

syntax

```
http.response.code=nnn
```

where *nnn* is a standard numeric range test with values in the range 100 to 999 inclusive.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, and `<Exception>` layers.
- Applies to HTTP transactions.

See Also

- Conditions: `http.request.version=`, `http.response.version=`
- Properties: `http.response.version( )`

http.response.version=

Tests the version of HTTP used by the origin server to deliver the response to the ProxySG.

Syntax

```
http.response.version=0.9|1.0|1.1
```

Layer and Transaction Notes

- Use in <Proxy>, <Cache>, and <Exception> layers.
- Applies to HTTP transactions.

See Also

- Conditions: http.request.version=, http.response.code=
- Properties: http.response.version( )

http.transparent_authentication=

This trigger evaluates to true if HTTP uses transparent proxy authentication for this request.

The trigger can be used with the `authenticate( )` or `authenticate.force( )` properties to select an authentication realm.

Syntax

```
http.transparent_authentication=yes|no
```

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, and `<Exception>` layers.
- Applies to HTTP transactions.

See Also

- Conditions: `attribute.`*`name`*`=`, `authenticated=`, `group=`, `has_attribute.name=`, `realm=`, `user=`, `user.domain=`
- Properties: `authenticate( )`, `authenticate.force( )`, `authenticate.mode( )`, `check_authorization( )`

http.x_method=

Tests HTTP request methods against any uncommon HTTP methods. A CPL parse warning is given if the method specified is a recognized method (in which case, `http.method=` is recommended).

Uncommon methods are tested using a string comparison, so some performance benefit exists with using `http.method=` when testing for common methods.

Syntax

```
http.x_method=method_name_list
```

where `http.x_method=` evaluates to NULL if the request is not an HTTP protocol request.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, and `<Exception>` layers.
- Applies to HTTP transactions.

See Also

- Conditions: `ftp.method=`, `http.method=`, `im.method=`, `method=`, `socks.method=`

im.buddy_id=

Tests the `buddy_id` associated with the instant messaging transaction.

Syntax

```
im.buddy_id[.case_sensitive]=user_id_string
im.buddy_id.substring[.case_sensitive]=substring
im.buddy_id.regex[.case_sensitive]="expr"
```

where:

- *`user_id_string`*—An exact match of the complete instant messaging buddy name.
- `substring` . . . *`substring`*—Specifies a substring of an instant messaging buddy name.
- `regex` . . . *`"expr"`*—Takes a regular expression.

Notes

- By default the test is case-insensitive. Specifying `.case_sensitive` makes the test case-sensitive.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Exception>` layers.
- Applies to instant messaging transactions.

See Also

- Actions: `append()`, `im.alert( )`, `set( )`
- Conditions: `im.chat_room.conference=`, `im.chat_room.id=`, `im.chat_room.invite_only=`, `im.chat_room.type=`, `im.chat_room.member=`, `im.chat_room.voice_enabled=`, `im.file.extension=`, `im.file.name=`, `im.file.path=`, `im.file.size=`, `im.message.route=`, `im.message.size=`, `im.message.text=`, `im.message.type=`, `im.method=`, `im.user_id=`
- Properties: `im.strip_attachments( )`

im.chat_room.conference=

Tests whether the chat room associated with the instant messaging transaction has the conference attribute set.

Syntax

```
im.chat_room.conference=yes|no
```

Layer and Transaction Notes

- Use in <Proxy> and <Exception> layers.
- Applies to instant messaging transactions.

See Also

- Actions: append(), im.alert( ), set( )
- Conditions: im.buddy_id=, im.chat_room.id=, im.chat_room.invite_only=, im.chat_room.type=, im.chat_room.member=, im.chat_room.voice_enabled=, im.file.extension=, im.file.name=, im.file.path=, im.file.size=, im.message.route=, im.message.size=, im.message.text=, im.message.type=, im.method=, im.user_id=
- Properties: im.strip_attachments( )

im.chat_room.id=

Tests the chat room ID associated with the instant messaging transaction.

Syntax

```
im.chat_room.id[.case_sensitive]=user_id_string
im.chat_room.id.substring[.case_sensitive]=substring
im.chat_room.id.regex[.case_sensitive]="expr"
```

where:

- *user_id_string*—An exact match of the complete chat room ID.
- substring . . . *substring*—Specifies a substring of a chat room ID.
- regex . . . *"expr"*—Takes a regular expression.

Notes

By default the test is case-insensitive. Specifying .case_sensitive makes the test case-sensitive.

Layer and Transaction Notes

- Use in <Proxy> and <Exception> layers.
- Applies to instant messaging transactions.

See Also

- Actions: append(), im.alert( ), set( )
- Conditions: im.buddy_id=, im.chat_room.conference=, im.chat_room.invite_only=, im.chat_room.type=, im.chat_room.member=, im.chat_room.voice_enabled=, im.file.extension=, im.file.name=, im.file.path=, im.file.size=, im.message.route=, im.message.size=, im.message.text=, im.message.type=, im.method=, im.user_id=
- Properties: im.strip_attachments( )

im.chat_room.invite_only=

Tests whether the chat room associated with the instant messaging transaction has the `invite_only` attribute set.

Syntax

```
im.chat_room.invite_only=yes|no
```

Layer and Transaction Notes

- Use in `<Proxy>` and `<Exception>` layers.
- Applies to instant messaging transactions.

See Also

- Actions: `append()`, `im.alert( )`, `set( )`
- Conditions: `im.buddy_id=`, `im.chat_room.conference=`, `im.chat_room.id=`, `im.chat_room.type=`, `im.chat_room.member=`, `im.chat_room.voice_enabled=`, `im.file.extension=`, `im.file.name=`, `im.file.path=`, `im.file.size=`, `im.message.route=`, `im.message.size=`, `im.message.text=`, `im.message.type=`, `im.method=`, `im.user_id=`
- Properties: `im.strip_attachments( )`

im.chat_room.type=

Tests whether the chat room associated with the transaction is public or private.

Syntax

```
im.chat_room.type=public|private
```

Layer and Transaction Notes

- Use in `<Proxy>` and `<Exception>` layers.
- Applies to instant messaging transactions.

See Also

- Actions: `append()`, `im.alert( )`, `set( )`
- Conditions: `im.buddy_id=`, `im.chat_room.conference=`, `im.chat_room.id=`, `im.chat_room.invite_only=`, `im.chat_room.member=`, `im.chat_room.voice_enabled=`, `im.file.extension=`, `im.file.name=`, `im.file.path=`, `im.file.size=`, `im.message.route=`, `im.message.size=`, `im.message.text=`, `im.message.type=`, `im.method=`, `im.user_id=`
- Properties: `im.strip_attachments( )`

im.chat_room.member=

Tests whether the chat room associated with the instant messaging transaction has a member matching the specified criterion.

Syntax

```
im.chat_room.id[.case_sensitive]=buddy_id_string
m.chat_room.id.substring[.case_sensitive]=substring
im.chat_room.id.regex[.case_sensitive]="expr"
```

where:

- *string*—An exact match of the complete instant messaging buddy ID.
- `substring . . .` *substring*—Specifies a substring of the instant messaging buddy ID.
- `regex . . .` *"expr"*—Takes a regular expression.

Notes

By default the test is case-insensitive. Specifying `.case_sensitive` makes the test case-sensitive.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Exception>` layers.
- Applies to instant messaging transactions.

See Also

- Actions: `append()`, `im.alert( )`, `set( )`
- Conditions: `im.buddy_id=`, `im.chat_room.conference=`, `im.chat_room.id=`, `im.chat_room.invite_only=`, `im.chat_room.type=`, `im.chat_room.voice_enabled=`, `im.file.extension=`, `im.file.name=`, `im.file.path=`, `im.file.size=`, `im.message.route=`, `im.message.size=`, `im.message.text=`, `im.message.type=`, `im.method=`, `im.user_id=`
- Properties: `im.strip_attachments( )`

im.chat_room.voice_enabled=

Tests whether the chat room associated with the instant messaging transaction is voice enabled.

Syntax

```
im.chat_room.voice_enabled=yes|no
```

Layer and Transaction Notes

- Use in <Proxy> and <Exception> layers.
- Applies to instant messaging transactions.

See Also

- Actions: `append()`, `im.alert( )`, `set( )`
- Conditions: `im.buddy_id=`, `im.chat_room.conference=`, `im.chat_room.id=`, `im.chat_room.invite_only=`, `im.chat_room.type=`, `im.chat_room.member=`, `im.file.extension=`, `im.file.name=`, `im.file.path=`, `im.file.size=`, `im.message.route=`, `im.message.size=`, `im.message.text=`, `im.message.type=`, `im.method=`, `im.user_id=`
- Properties: `im.strip_attachments( )`

im.file.extension=

Tests the file extension of a file associated with an instant messaging transaction. The leading '.' of the file extension is optional. Only supports an exact match.

Syntax

```
im.file.extension[.case-sensitive]=[.]filename_extension
```

Notes

By default the test is case-insensitive. Specifying `.case_sensitive` makes the test case-sensitive.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Exception>` layers.
- Applies to instant messaging transactions.

See Also

- Actions: `append()`, `im.alert( )`, `set( )`
- Conditions: `im.buddy_id=`, `im.chat_room.conference=`, `im.chat_room.id=`, `im.chat_room.invite_only=`, `im.chat_room.type=`, `im.chat_room.member=`, `im.chat_room.voice_enabled=`, `im.file.name=`, `im.file.path=`, `im.file.size=`, `im.message.route=`, `im.message.size=`, `im.message.text=`, `im.message.type=`, `im.method=`, `im.user_id=`
- Properties: `im.strip_attachments( )`

im.file.name=

Tests the file name (the last component of the path), including the extension, of a file associated with an instant messaging transaction.

Syntax

```
im.file.name[.case_sensitive]=string
im.file.name.prefix[.case_sensitive]=prefix_string
im.file.name.substring[.case_sensitive]=substring
im.file.name.regex[.case_sensitive]="expr"
```

where:

- *string*—An exact match of the complete file name with extension.
- `prefix` . . . *prefix_string*—Specifies a prefix match.
- `substring` . . . *substring*—Specifies a substring match of the file name.
- `regex` . . . *"expr"*—Takes a regular expression.

Notes

By default the test is case-insensitive. Specifying `.case_sensitive` makes the test case-sensitive.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Exception>` layers.
- Applies to instant messaging transactions.

See Also

- Actions: `append( )`, `im.alert( )`, `set( )`
- Conditions: `im.buddy_id=`, `im.chat_room.conference=`, `im.chat_room.id=`, `im.chat_room.invite_only=`, `im.chat_room.type=`, `im.chat_room.member=`, `im.chat_room.voice_enabled=`, `im.file.extension=`, `im.file.path=`, `im.file.size=`, `im.message.route=`, `im.message.size=`, `im.message.text=`, `im.message.type=`, `im.method=`, `im.user_id=`
- Properties: `im.strip_attachments( )`

im.file.path=

Tests the file path of a file associated with an instant messaging transaction against the specified criterion.

Syntax

```
im.file.path[.case_sensitive]=string
im.file.path.prefix[.case_sensitive]=prefix_string
im.file.path.substring[.case_sensitive]=substring
im.file.path.regex[.case_sensitive]="expr"
```

where:

- *string*—An exact match of the complete path.
- `prefix` . . . *prefix_string*—Specifies a prefix match.
- `substring` . . . *substring*—Specifies a substring match of the path.
- `regex` . . . *"expr"*—Takes a regular expression.

Notes

By default the test is case-insensitive. Specifying `.case_sensitive` makes the test case-sensitive.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Exception>` layers.
- Applies to instant messaging transactions.

See Also

- Actions: `append()`, `im.alert( )`, `set( )`
- Conditions: `im.buddy_id=`, `im.chat_room.conference=`, `im.chat_room.id=`, `im.chat_room.invite_only=`, `im.chat_room.type=`, `im.chat_room.member=`, `im.chat_room.voice_enabled=`, `im.file.extension=`, `im.file.name=`, `im.file.size=`, `im.message.route=`, `im.message.size=`, `im.message.text=`, `im.message.type=`, `im.method=`, `im.user_id=`
- Properties: `im.strip_attachments( )`

im.file.size=

Performs a signed 64-bit range test of the size of a file associated with an instant messaging transaction.

Syntax

```
im.file.size=[min]..[max]
```

The default minimum value is zero (`0`); there is no default maximum value.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Exception>` layers.
- Applies to instant messaging transactions.

See Also

- Actions: `append()`, `im.alert( )`, `set( )`
- Conditions: `im.buddy_id=`, `im.chat_room.conference=`, `im.chat_room.id=`, `im.chat_room.invite_only=`, `im.chat_room.type=`, `im.chat_room.member=`, `im.chat_room.voice_enabled=`, `im.file.extension=`, `im.file.name=`, `im.file.path=`, `im.message.route=`, `im.message.size=`, `im.message.text=`, `im.message.type=`, `im.method=`, `im.user_id=`
- Properties: `im.strip_attachments( )`

im.message.opcode=

Tests the value of an opcode associated with an instant messaging transaction whose `im.method` is `send_unknown` or `receive_unknown`.

Note: Generally, this is used with `deny( )` to restrict interactions that are new to one of the supported instant messaging protocols and for which direct policy control is not yet available. Use of this trigger requires specific values for the opcode as determined by Blue Coat Systems technical support.

Syntax

```
im.message.opcode=string
```

where *`string`* is a value specified by technical support.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Exception>` layers.
- Applies to instant messaging transactions.

im.message.route=

Tests how the instant messaging message reaches its recipients.

Syntax

```
im.message.route=service|direct|chat
```

where:

- `service`—The message is relayed through the IM service.
- `direct`—The message is sent directly to the recipient.
- `chat`—The message is sent to a chat room (includes conferences).

Layer and Transaction Notes

- Use in `<Proxy>` and `<Exception>` layers.
- Applies to instant messaging transactions.

See Also

- Actions: `append()`, `im.alert( )`, `set( )`
- Conditions: `im.buddy_id=`, `im.chat_room.conference=`, `im.chat_room.id=`, `im.chat_room.invite_only=`, `im.chat_room.type=`, `im.chat_room.member=`, `im.chat_room.voice_enabled=`, `im.file.extension=`, `im.file.name=`, `im.file.path=`, `im.file.size=`, `im.message.size=`, `im.message.text=`, `im.message.type=`, `im.method=`, `im.user_id=`
- Properties: `im.strip_attachments( )`

im.message.size=

Performs a signed 64-bit range test on the size of the instant messaging message.

Syntax

```
im.message.size=[min]..[max]
```

The default minimum value is zero (0); there is no default maximum value.

Layer and Transaction Notes

- Use in <Proxy> and <Exception> layers.
- Applies to instant messaging transactions.

See Also

- Actions: append(), im.alert( ), set( )
- Conditions: im.buddy_id=, im.chat_room.conference=, im.chat_room.id=, im.chat_room.invite_only=, im.chat_room.type=, im.chat_room.member=, im.chat_room.voice_enabled=, im.file.extension=, im.file.name=, im.file.path=, im.file.size=, im.message.route=, im.message.text=, im.message.type=, im.method=, im.user_id=
- Properties: im.strip_attachments( )

im.message.text=

Tests if the message text contains the specified text or pattern.

Note: The `.regex` version of this test is limited to the first 8K of the message. The `.substring` version of the test does not have this restriction.

Syntax

```
im.message.text.substring[.case_sensitive]=substring
im.message.text.regex[.case_sensitive]=expr
```

where:

- `substring` . . . *`substring`*—Specifies a substring match of the message text.
- `regex` . . . *`"expr"`*—Takes a regular expression.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Exception>` layers.
- Applies to instant messaging transactions.

See Also

- Actions: `append()`, `im.alert( )`, `set( )`
- Conditions: `im.buddy_id=`, `im.chat_room.conference=`, `im.chat_room.id=`, `im.chat_room.invite_only=`, `im.chat_room.type=`, `im.chat_room.member=`, `im.chat_room.voice_enabled=`, `im.file.extension=`, `im.file.name=`, `im.file.path=`, `im.file.size=`, `im.message.route=`, `im.message.size=`, `im.message.type=`, `im.method=`, `im.user_id=`
- Properties: `im.strip_attachments( )`

im.message.type=

Tests the message type of an instant messaging transaction.

Syntax

```
im.message.type=text|invite|voice_invite|file|file_list|application
```

where:

- `text`—Normal IM text message.
- `invite`—An invitation to a chat room or to communicate directly.
- `voice_invite`—Invitation to a voice chat.
- `file`—The message contains a file.
- `file_list`—The message contains a list of exported files.
- `application`—Tests if this instant messaging request was generated internally by the instant messaging application, rather than as a direct result of a user gesture.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Exception>` layers.
- Applies to instant messaging transactions.

See Also

- Actions: `append()`, `im.alert( )`, `set( )`
- Conditions: `im.buddy_id=`, `im.chat_room.conference=`, `im.chat_room.id=`, `im.chat_room.invite_only=`, `im.chat_room.type=`, `im.chat_room.member=`, `im.chat_room.voice_enabled=`, `im.file.extension=`, `im.file.name=`, `im.file.path=`, `im.file.size=`, `im.message.route=`, `im.message.size=`, `im.message.text=`, `im.method=`, `im.user_id=`
- Properties: `im.strip_attachments( )`

im.method=

Tests the method associated with the instant messaging transaction.

Syntax

```
im.method=open|create|join|join_user|login|logout|notify_join|notify_quit|
notify_state|quit|receive|receive_unknown|send|send_unknown|set_state
```

Layer and Transaction Notes

- Use in <Proxy>, <Cache>, and <Exception> layers.
- Applies to instant messaging transactions.

See Also

- Actions: `append()`, `im.alert( )`, `set( )`
- Conditions: `ftp.method=`, `http.method=`, `http.x_method=`, `method=`, `socks.method=`
- IM Conditions: `im.buddy_id=`, `im.chat_room.conference=`, `im.chat_room.id=`, `im.chat_room.invite_only=`, `im.chat_room.type=`, `im.chat_room.member=`, `im.chat_room.voice_enabled=`, `im.file.extension=`, `im.file.name=`, `im.file.path=`, `im.file.size=`, `im.message.route=`, `im.message.size=`, `im.message.text=`, `im.message.type=`, `im.user_id=`
- Properties: `im.strip_attachments( )`

im.user_id=

Tests the *user_id* associated with the instant messaging transaction.

Syntax

```
im.user_id[.case_sensitive]=user_id_string
im.user_id.substring[.case_sensitive]=substring
im.user_id.regex[.case_sensitive]=“expr”
```

where:

- *user_id_string*—An exact match of the complete instant messaging username.
- `substring ...` *substring*—Specifies a substring of an instant messaging username.
- `regex ...` *"expr"*—Takes a regular expression.

Notes

By default the test is case-insensitive. Specifying `.case_sensitive` makes the test case-sensitive.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Exception>` layers.
- Applies to instant messaging transactions.

See Also

- Conditions: `im.buddy_id=`, `im.chat_room.conference=`, `im.chat_room.id=`, `im.chat_room.invite_only=`, `im.chat_room.type=`, `im.chat_room.member=`, `im.chat_room.voice_enabled=`, `im.file.extension=`, `im.file.name=`, `im.file.path=`, `im.file.size=`, `im.message.route=`, `im.message.size=`, `im.message.text=`, `im.message.type=`, `im.method=`
- Properties: `im.strip_attachments( )`
- Actions: `append( )`, `im.alert( )`, `set( )`

live=

Tests if the streaming content is a live stream.

Syntax

```
live=yes|no
```

Layer and Transaction Notes

- Use in <Cache> and <Proxy> layers.
- Applies to streaming transactions.

Examples

```
; The following policy restricts access to live streams during morning hours.
; In this example, we use a policy layer to define policy just for the live streams.
; This example uses the restrict form and integrates with other <proxy> layers.

<proxy>
   deny live=yes time=1200..0800 ; Policy for live streams
```

See Also

- Conditions: bitrate=, streaming.client=, streaming.content=
- Properties: access_server( ), max_bitrate(), streaming.transport( )

method=

Tests the protocol method name associated with the transaction. Appropriate method names depend on the protocol. Also, a warning is issued during policy file compilation if the name is not a recognized method.

`method=` accepts any of the protocol specific methods accepted by `admin.access=`, `ftp.method=`, `http.method=`, `im.method=`, or `socks.method=`.

It also recognizes `ICP_QUERY`, `MMS_PLAY`, and `RTSP_PLAY`.

It accepts, but gives a parse warning for, unrecognized methods.

Matches are done by case insensitive string comparison, so there is a performance benefit to using protocol specific tests, in addition to the extra error checking available.

A specified method can match a commonly named method from multiple protocols (for example, CONNECT).

Note: Use of `method=` in `<Admin>` layers has been replaced by `admin.access=`.

Syntax

```
method=method_name
```

where *`method_name`* is a valid method appropriate for the protocol of interest. Method names are case-insensitive. The following methods are recognized:

Protocol	Methods
HTTP, HTTPS See "http.method=" on page 79, "http.x_method=" on page 84	`TUNNEL` `GET`, `HEAD`, `POST`, `PUT`, `CONNECT`, `DELETE`, `OPTIONS`, `TRACE` (HTTP 1.1/rfc 2616) `PROPFIND PROPPATCH MKCOL COPY MOVE LOCK UNLOCK` (WebDAV/rfc 2518) `MKDIR INDEX RMDIR COPY MOVE` (Netscape) `LINK UNLINK PATCH` (HTTP 1.1/rfc 2068, dropped in rfc 2616)
FTP See "ftp.method=" on page 71.	`ABOR ACCT ALLO APPE CDUP CWD DELE HELP LIST MDTM MKD MODE NLST NOOP PASS PASV PORT PWD RETR RMD RNFR RNTO SITE SIZE SMNT STOR STOU STRU SYST TYPE USER XCUP XCWD XMKD XPWD XRMD`
ICP	`ICP_QUERY`
instant messaging	`OPEN`, `CREATE`, `JOIN`, `JOIN_USER`, `LOGIN`, `LOGOUT`, `NOTIFY_JOIN`, `NOTIFY_QUIT`, `NOTIFY_STATE`, `QUIT`, `RECEIVE`, `RECEIVE_UNKNOWN`, `SEND`, `SEND_UNKNOWN`, `SET_STATE`
Real Media (RTSP)	`RTSP_PLAY`
SOCKS	`CONNECT`, `BIND`, `UDP_ASSOCIATE`
Windows Media (MMS)	`MMS_PLAY`
Windows Media HTTP streaming (HTTP then MMS)	`GET` then `MMS_PLAY`

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, and `<Exception>` layers.

Examples

```
<proxy>
   http.method=GET response.header.Pragma=”no-cache" deny


; This example is applicable to a blacklist model. It denies access to
; transparent FTP by denying the OPEN method on port 21.
<proxy> proxy.port=21
   deny ftp.method=OPEN


; This example tests method=CONNECT to secure against firewall bypass
<proxy>
   deny method=CONNECT server_url.port=!443
```

See Also

- Conditions: `admin.access=`, `category=`, `console_access=`, `content_management=`, `ftp.method=`, `http.method=`, `http.x_method=`, `im.method=`, `server_url=`, `socks.method=`
- Properties: `http.request.version( )`, `http.response.version( )`

minute=

Tests if the minute of the hour is in the specified range or an exact match. By default, the ProxySG appliance's clock and time zone are used to determine the current minute. To specify the UTC time zone, use the form `minute.utc=`. The numeric pattern used to test the `minute` condition can contain no whitespace.

Syntax

```
minute[.utc]={[first_minute]..[last_minute]|exact_minute}
```

where:

- *`first_minute`*—An integer from 0 to 59, indicating the first minute of the hour that tests true. If left blank, minute 0 is assumed.
- *`last_minute`*—An integer from 0 to 59, indicating the last minute of the hour that tests true. If left blank, minute 59 is assumed.
- *`exact_minute`*—An integer from 0 to 59, indicating the minute of each hour that tests true.

Note: To test against an inverted range, such as a range that crosses from one hour into the next, the following shorthand expression is available. While `minute=(..14|44..)` specifies the first 15 minutes and last 15 minutes of each hour, the policy language also recognizes `minute=44..14` as equivalent.

Layer and Transaction Notes

- Use in all layers.
- Using time-related conditions to control caching behavior in a `<Cache>` layer may cause thrashing of the cached objects.

Examples

```
; Tests for the first 5 minutes of every hour.
minute=0..4
```

See Also

- Conditions: `date[.utc]=`, `day=`, `hour=`, `month=`, `time=`, `weekday=`, `year=`

month=

Tests if the month is in the specified range or an exact match. By default, the Proxy*SG* appliance's date and time zone are used to determine the current month. To specify the UTC time zone, use the form `month.utc=`. The numeric pattern used to test the `month` condition can contain no whitespace.

Syntax

```
month[.utc]={[first_month]..[last_month]|exact_month}
```

where:

- *`first_month`*—An integer from 1 to 12, where 1 specifies January and 12 specifies December, specifying the first month that tests true. If left blank, January (month 1) is assumed.
- *`last_month`*—An integer from 1 to 12, where 1 specifies January and 12 specifies December, specifying the last month that tests true. If left blank, December (month 12) is assumed.
- *`exact_month`*—An integer from 1 to 12, where 1 specifies January and 12 specifies December, indicating the month that tests true.

Note: To test against an inverted range, such as a range that crosses from one year into the next, the following shorthand expression is available. While `month=(..6|9..)` specifies September through June, the policy language also recognizes `month=9..6` as equivalent.

Layer and Transaction Notes

- Use in all layers.
- Using time-related conditions to control caching behavior in a `<Cache>` layer may cause thrashing of the cached objects.

Examples

```
; Tests for the year-end holiday season.

define condition year_end_holidays
month=12 day=25..
month=1 day=1
end_condition year_end_holidays
```

See Also

- Conditions: `date[.utc]=`, `day=`, `hour=`, `minute=`, `time=`, `weekday=`, `year=`

protocol=

The `protocol=` condition has been deprecated in favor of `url.scheme=`. For more information see "url=" on page 137.

See Also

Conditions: `client.protocol=`

proxy.address=

Tests the destination address of the arriving IP packet. The expression can include an IP address or subnet, or the label of a subnet definition block.

If the transaction was explicitly proxied, then `proxy.address=` tests the IP address the client used to reach the proxy, which is either the IP address of the NIC on which the request arrived or a virtual IP address. This is intended for situations where the proxy has a range of virtual IP address; you can use `proxy.address=` to test which virtual IP address was used to reach the proxy.

If the transaction was transparently proxied, then `proxy.address=` tests the destination address contained in the IP packet. Note that this test may not be equivalent to testing the `server_url.address`. The `server_url.address` and `proxy.address` conditions test different addresses in the case where a proxied request is transparently intercepted: `server_url.address=` contains the address of the origin server, and `proxy.address=` contains the address of the upstream proxy through which the request is to be handled.

Note: `proxy.card=` functions correctly for transparent transactions.

Replaces: `proxy_address=`

Syntax

```
proxy.address=ip_address|subnet|subnet_label
```

where:

- *`ip_address`*—NIC IP address or subnet; for example, `10.1.198.54`.
- *`subnet`*—A subnet mask; for example, `10.1.198.0/24`
- *`subnet_label`*—Label of a subnet definition block that binds a number of IP addresses or subnets.

Layer and Transaction Notes

- Use in `<Admin>`, `<Proxy>`, and `<Forward>` layers.
- Applies to proxy transactions.

Examples

```
; Service should be denied through proxy within the subnet 1.2.3.x.
<proxy>
    proxy.address=1.2.3.0/24 deny
```

See Also

- Conditions: `client.address=`, `client.protocol=`, `proxy.card=`, `proxy.port=`
- Definitions: `define subnet`

proxy.card=

Tests the ordinal number of the network interface card (NIC) used by a request.

Replaces: `proxy_card`

Syntax

```
proxy.card=card_number
```

where *card_number* is an integer that reflects the installation order.

Layer and Transaction Notes

- Use in <Admin>, <Proxy>, and <Forward> layers.
- Applies to proxy transactions.

Examples

```
; Deny all incoming traffic through proxy card 0.
<proxy>
   proxy.card=0 deny
```

See Also

- Conditions: `client.address=`, `client.protocol=`, `proxy.address=`, `proxy.port=`

proxy.port=

Tests if the IP port used by a request is within the specified range or an exact match.The numeric pattern used to test the `proxy.port=` condition can contain no whitespace.

If the transaction was explicitly proxied, then this tests the IP port that the client used to reach the proxy. The pattern is a number between 1 and 65535 or a numeric range.

If the transaction was transparently proxied, however, then `proxy.port=` tests which port the client thinks it is connecting to on the upstream proxy device or origin server. If the client thinks it is connecting directly to the origin server, but is transparently proxied, and if the port number specified by the client in the request URL is not inconsistent or falsified, then `proxy.port=` and `server_url.port=` are testing the same value.

Note: Since the Proxy*SG* default configuration passes through tunneled traffic, some changes must be made to begin transparent port monitoring. Only proxy ports that have been configured and enabled can be tested using the `proxy.port=` condition. For example, if the transparent FTP service, on port 21, is either not configured or not enabled, a policy rule that includes `proxy.port=21` has no effect.

Replaces: `proxy_port=`

Syntax

```
proxy.port={[low_port_number]..[high_port_number]|exact_port_number}
```

where:

- *`low_port_number`*—A port number at the low end of the range to be tested. Can be a number between 1 and 65535.
- *`high_port_number`*—A port number at the high end of the range to be tested. Can be a number between 1 and 65535.
- *`exact_port_number`*—A single port number; for example, `80`. Can be a number between 1 and 65535.

Layer and Transaction Notes

- Use in `<Admin>`, `<Proxy>`, and `<Forward>` layers.
- Applies to proxy transactions.

Examples

```
; Deny URL through the default proxy port.
<proxy>
    url=http://www.example.com proxy.port=8080 deny
```

See Also

- Conditions: `client.address=`, `client.protocol=`, `proxy.address=`, `proxy.card=`, `proxy.port=`, `server_url.port=`

realm=

Tests if the client is authenticated and if the client has logged into the specified realm. If both of these conditions are met, the response is true. In addition, the `group=` condition can be used to test whether the user belongs to the specified group. This trigger is unavailable if the current transaction is not authenticated (for example, the `authenticate` property is set to `no`).

If you reference more than one realm in your policy, consider disambiguating user, group and attribute tests by combining them with a `realm=test`. This reduces the number of extraneous queries to authentication services for group, user or attribute information that does not pertain to that realm.

Syntax

```
realm=realm_name
```

where *realm_name* is the name of an NTLM, Local Password, RADIUS, LDAP, Certificate, or Sequence realm. Realm names are case-insensitive for all realm types.

Layer and Transaction Notes

- Use in `<Admin>` and `<Proxy>` layers.
- Applies to proxy and administrator transactions.

Examples

```
; This example tests if the user has logged into realm corp and
; is authenticated in the specified group.

realm=corp group=all_staff


; This example uses the realm property to distinguish the policy applied
; to two groups of users--corp's employees, and their corporate partners and
; clients. These two groups will authenticate in different realms.

<proxy>
   client.address=10.10.10/24 authenticate(corp)   ; The corporate realm
      authenticate(client) ; Company partners & clients

<proxy> realm=corp ; Rules for corp employees
   allow url.domain=corp.com ; Unrestricted internal access
      category=(violence, gambling) exception(content_filter_denied)

<proxy> realm=client ; Rules for business partners & clients
   allow group=partners url=corp.com/partners ; Restricted to partners
   allow group=(partners, clients) url=corp.com/clients ; Both groups allowed
      deny

; Additional layers would continue to be guarded with the realm, so that only
; the 'client' realm would be queried about the 'partners' and 'clients' groups.
```

See Also

- Conditions: `attribute.`*name*`=`, `authenticated=`, `group=`, `has_attribute.`*name*`=`, `http.transparent_authentication=`, `user=`, `user.domain=`

- Properties: `authenticate( )`, `authenticate.force( )`, `check_authorization( )`

release.id=

Tests the release ID of the Proxy*SG* software. The release ID of the Proxy*SG* software currently running is displayed on the main page of the Management Console and in the Management>Maintenance>Upgrade>Systems tab of the Management Console. It also can be displayed through the CLI using the `show version` command.

Replaces: `release_id=`

Syntax

```
release.id=number
```

where *number* is a five-digit number that increases with each new release of Proxy*SG*.

Layer and Transaction Notes

- May be used in any type of layer.

Examples

```
; the condition below is only true if you are running a version of ProxySG
; whose release id is 18000 or later

release.id=18000..
```

See Also

- Conditions: `release.version=`

release.version=

Tests the release version of the Proxy*SG* software. The release version of the Proxy*SG* software currently running is displayed on the main page of the Management Console and in the Management>Maintenance>Upgrade>Systems tab of the Management Console. It also can be displayed through the CLI using the `show version` command.

Replaces: `release_version=`

Syntax

```
release.version={[minimum_version]..[maximum_version]|version}
```

where each of the versions is of the format:.

```
major_#.minor_#.dot_#.patch_#
```

Each number must be in the range 0 to 255. The *major_#* is required; less significant portions of the version may be omitted and will default to 0.

Layer and Transaction Notes

- May be used in any layer.

Examples

```
; the condition below is only true if you are running a version of ProxySG
; whose release version is 3.1. or greater

release.version=3.1...

; the condition below is only true if you are running a version of ProxySG
; whose release version is less or equal to than 3.1.2

release.version=..3.1.2
```

request.header.*header_name*=

Tests the specified request header (*header_name*) against a regular expression. Any recognized HTTP request header can be tested. For custom headers, use `request_x_header.`*`header_name`*`=` instead. For streaming requests, only the `User-Agent` header is available.

Replaces: `request_header.`*`header_name`*`=`

Syntax

```
request.header.header_name=regular_expression
```

where:

- *`header_name`*—A recognized HTTP header. For a complete list of recognized headers, see Appendix C: "Recognized HTTP Headers".
- *`regular_expression`*—A regular expression. For more information, refer to Appendix E: "Using Regular Expressions," in the Blue Coat ProxySG *Configuration and Management Guide*.

Layer and Transaction Notes

- Use in `<Cache>` and `<Proxy>` layers.

Examples

```
;deny access when request is sent with Pragma-no-cache header
<proxy>
    deny url=http://www.bluecoat.com request.header.Pragma="no-cache"
```

See Also

- Actions: `append( )`, `delete( )`, `delete_matching( )`, `rewrite( )`, `set( )`
- Conditions: `request.header.`*`header_name`*`.address=`, `request.x_header.`*`header_name`*`=`, `response.header.`*`header_name`*`=`

request.header.*header_name*.address=

Tests if the specified request header can be parsed as an IP address; otherwise, false. If parsing succeeds, then the IP address extracted from the header is tested against the specified IP address. The expression can include an IP address or subnet, or the label of a subnet definition block. The header must be a common HTTP header. This condition is commonly used with the `X-Forwarded-For` and `Client-IP` headers. For other, custom headers, use `request.x_header.`*`header_name`*`.address=`.

Replaces: `request_header_address.`*`header_name`*`=`

Syntax

`request.header.`*`header_name`*`.address=`*`ip_address`*`|`*`subnet`*`|`*`subnet_label`*

where:

- *`header_name`*—A recognized HTTP header. For a complete list of recognized headers, see Appendix C: "Recognized HTTP Headers".
- *`ip_address`*—IP address; for example, `10.1.198.46`.
- *`subnet`*—A subnet mask; for example, `10.1.198.0/24`.
- *`subnet_label`*—Label of a subnet definition block that binds a number of IP addresses or subnets.

Layer and Transaction Notes

- Use in `<Cache>` and `<Proxy>` layers.

Examples

```
; In this example, we assume that there is a downstream ProxySG that
; identifies client traffic by putting the client's IP address in a request
; header.

; Here we'll deny access to some clients, based on the header value.

<proxy>

       ; Netscape's convention is to use the Client-IP header

       deny request.header.Client-IP.address=10.1.198.0/24 ; the subnet

   ; Blue Coat's convention is to use the extended header:

       deny request.header.X-Forwarded-For.address=10.1.198.12
```

See Also

- Actions: `append( )`, `delete( )`, `delete_matching( )`, `rewrite( )`, `set( )`
- Conditions: `request.header.`*`header_name`*`=`, `response.header.`*`header_name`*`=`, `response.x_header.`*`header_name`*`=`
- Definitions: `define subnet`

request.header.Referer.url=

Test if the URL specified by the Referer header matches the specified criteria. The basic `request.header.Referer.url=` test attempts to match the complete Referer URL against a specified pattern. The pattern may include the scheme, host, port, path and query components of the URL. If any of these is not included in the pattern, then the corresponding component of the URL is not tested and can have any value.

Specific portions of the Referer URL can be tested by applying URL component modifiers to the trigger. In addition to component modifiers, optional test type modifiers can be used to change the way the pattern is matched.

This trigger is unavailable if the Referer header is missing, or if its value cannot be parsed as a URL. If the Referer header contains a relative URL, the requested URL is used as a base to form an absolute URL prior to testing.

Syntax

```
request.header.Referer.url[.case_sensitive][.no_lookup]=prefix_pattern
request.header.Referer.url.domain[.case_sensitive][.no_lookup]=
                                                        domain_suffix_pattern
request.header.Referer.url.regex[.case_sensitive]=regular_expression

request.header.Referer.url.address=ip_address|subnet|subnet_label
request.header.Referer.url.extension[.case_sensitive]=[.]filename_extension

request.header.Referer.url.host[.exact][.no_lookup]=host
request.header.Referer.url.host.[prefix|substring|suffix][.no_lookup]=string
request.header.Referer.url.host.is_numeric=yes|no
request.header.Referer.url.host.no_name=yes|no

request.header.Referer.url.path[.case_sensitive]=/string
request.header.Referer.url.path[.substring|.suffix][.case_sensitive]=string
request.header.Referer.url.path.regex[.case_sensitive]=regular_expression

request.header.Referer.url.port={[low_port_number]..[high_port_number]
                                 |exact_port_number}

request.header.Referer.url.query.regex[.case_sensitive]=regular_expression

request.header.Referer.url.scheme=url_scheme
```

where all options are identical to `url=`, except for the URL being tested. For more information, see "url=" on page 137.

Discussion

The `request.header.Referer.url=` condition is identical to `url=`, except for the lack of a `define url` condition and `[url]` or `[url.domain]` sections.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, and `<Exception>` layers.
- Applies to HTTP proxy transactions.

Examples

```
; Test if the Referer URL includes this pattern, and block access.
```

```
; Relative URLs, such as docs subdirectories and pages, will match.
deny request.header.Referer.url=http://www.example.com/docs

; Test if the Referer URL host's IP address is a match.
request.header.Referer.url.address=10.1.198.0

; Test whether the Referer URL includes company.com as domain.
request.header.Referer.url.domain=company.com

; Test whether the Referer URL includes .com.
request.header.Referer.url.domain=.com

; Test if the Referer URL includes this domain-suffix pattern,
; and block service. Relative URLs, such as docs
; subdirectories and pages, will match.
deny request.header.Referer.url.domain=company.com/docs

; examples of the use of request.header.Referer.url.extension=
request.header.Referer.url.extension=.txt
request.header.Referer.url.extension=(.htm, .html)
request.header.Referer.url.extension=(img, jpg, jpeg)

; This example matches the first Referer header value and doesn't match the second
from
; the following two requests:
; 1) Referer: http://1.2.3.4/test
; 2) Referer: http://www.example.com
<proxy>
request.header.Referer.url.host.is_numeric=yes

; In the example below we assume that 1.2.3.4 is the IP of the host mycompany
; The condition will match the following two requests if the reverse DNS was
; successful:
; 1) Referer: http://1.2.3.4/
; 2) Referer: http://mycompany.com/
; If the reverse DNS fails then the first request is not matched
```

```
<proxy>
request.header.Referer.url.host.regex=mycompany


; request.header.Referer.url.path tests
; The following request.header.Referer.url.path strings would all match the example
Referer URL:
; Referer: http://www.example.com/cgi-bin/query.pl?q=test#fragment
request.header.Referer.url.path="/cgi-bin/query.pl?q=test"
request.header.Referer.url.path="/cgi-bin/query.pl"
request.header.Referer.url.path="/cgi-bin/"
request.header.Referer.url.path="/cgi" ; partial components match too
request.header.Referer.url.path="/" ; Always matches regardless of URL.


; Testing the Referer URL port
request.header.Referer.url.port=80
```

See Also

- Conditions: `url=, server_url=`
- Definitions: `define subnet`

request.x_header.*header_name*=

Tests the specified request header (*header_name*) against a regular expression. Any HTTP request header can be tested, including custom headers. To test recognized headers, use `request.header.`*header_name*`=` instead, so that typing errors can be caught at compile time. For streaming requests, only the `User-Agent` header is available.

Replaces: `request_x_header.`*header_name*`=`

Syntax

```
request.x_header.header_name=regular_expression
```

where:

- *header_name*—Any HTTP header, including custom headers.
- *regular_expression*—A regular expression. For more information, see Appendix E: "Using Regular Expressions," in the Blue Coat Proxy*SG Configuration and Management Guide*.

Layer and Transaction Notes

- Use in `<Cache>` and `<Proxy>` layers.

Examples

```
; deny access to the URL below if the request contains the custom
; header "Test" and the header has a value of "test1"

<proxy>
    deny url=http://www.bluecoat.com request.x_header.Test="test1"
```

See Also

- Actions: `append( )`, `delete( )`, `delete_matching( )`, `rewrite( )`, `set( )`
- Conditions: `request.header.`*header_name*`=`, `request.header.`*header_name*`.address=`, `response.x_header.`*header_name*`=`

request.x_header.*header_name*.address=

Tests if the specified request header can be parsed as an IP address; otherwise, false. If parsing succeeds, then the IP address extracted from the header is tested against the specified IP address. The expression can include an IP address or subnet, or the label of a subnet definition block. This condition is intended for use with custom headers other than `X-Forwarded-For` and `Client-IP` headers; for these, use `request.header.`*`header_name`*`.address=` so that typing any errors can be caught at compile time.

Replaces: `request_x_header.`*`header_name`*`.address=`

Syntax

```
request.x_header.header_name.address= ip_address|subnet|subnet_label
```

where:

- *`header_name`*—Any HTTP header, including custom headers.
- *`ip_address`*—IP address; for example, `10.1.198.0`.
- *`subnet`*—A subnet mask; for example, `10.1.198.0/24`.
- *`subnet_label`*—Label of a subnet definition block that binds a number of IP addresses or subnets.

Layer and Transaction Notes

- Use in `<Cache>` and `<Proxy>` layers.

Examples

```
; deny access if the request's custom header "Local" has the value 10.1.198.0
deny request.x_header.Local.address=10.1.198.0
```

See Also

- Actions: `append( )`, `delete( )`, `delete_matching( )`, `rewrite( )`, `set( )`
- Conditions: `request.header.`*`header_name`*`=`, `request.header.`*`header_name`*`.address=`, `response.x_header.`*`header_name`*`=`
- Definitions: `define subnet`

response.header.*header_name*=

Tests the specified response header (*header_name*) against a regular expression. Any recognized HTTP response header can be tested. For custom headers, use `response.x_header.`*header_name*= instead.

Replaces: `response_header.`*header_name*=

Syntax

```
response.header.header_name=regular_expression
```

where:

- *header_name*—A recognized HTTP header. For a list of recognized headers, see Appendix C: "Recognized HTTP Headers". For custom headers not listed, use condition `response.x_header.`*header_name* instead.
- *regular_expression*—A regular expression. For more information, refer to Appendix E: "Using Regular Expressions," in the Proxy*SG Configuration and Management Guide*.

Layer and Transaction Notes

- Use in `<Cache>`, `<Proxy>`, and `<Exception>` layers.

Examples

```
; Test if the response's "Content-Type" header has the value "image/jpeg"
response.header.Content-Type="image/jpeg"
```

See Also

- Actions: `append( )`, `delete( )`, `delete_matching( )`, `rewrite( )`, `set( )`
- Conditions: `request.header.`*header_name*=, `response.x_header.`*header_name*=

response.x_header.*header_name=*

Tests the specified response header (*header_name*) against a regular expression. For HTTP requests, any response header can be tested, including custom headers. For recognized HTTP headers, use `response.header.`*header_name*`=` instead so that typing errors can be caught at compile time.

Replaces: `response_x_header.`*header_name*`=`

Syntax

```
response.x_header.header_name=regular_expression
```

where:

- *header_name*—Any HTTP header, including custom headers.
- *regular_expression*—A regular expression. For more information, see Appendix E: "Using Regular Expressions," in the Blue Coat ProxySG *Configuration and Management Guide*.

Layer and Transaction Notes

- Use in `<Cache>`, `<Proxy>`, and `<Exception>` layers.

Examples

```
; Tests if the custom header "Security" has the value of "confidential"
response.x_header.Security="confidential"
```

See Also

- Actions: `append( )`, `delete( )`, `delete_matching( )`, `rewrite( )`, `set( )`
- Conditions: `request.x_header.`*header_name=*, `response.header.`*header_name=*

server_url=

Tests if a portion of the URL used in server connections matches the specified criteria. The basic `server_url=` test attempts to match the complete possibly-rewritten request URL against a specified pattern. The pattern may include the scheme, host, port, path and query components of the URL. If any of these is not included in the pattern, then the corresponding component of the URL is not tested and can have any value.

Specific portions of the URL can be tested by applying URL component modifiers to the trigger. In addition to component modifiers, optional test type modifiers can be used to change the way the pattern is matched.

Note: This set of tests match against the requested URL, taking into account the effect of any `rewrite( )` actions. Because any rewrites of the URL intended for servers or other upstream devices must be respected by `<Forward>` layer policy, the `url=` triggers are not allowed in `<Forward>` layers. Instead, the equivalent set of `server_url=` tests are provided for use in the `<Forward>` layer. Those tests always take into account the effect of any `rewrite( )` actions on the URL.

Syntax

```
server_url[.case_sensitive][.no_lookup]=prefix_pattern
server_url.domain[.case_sensitive][.no_lookup]=domain_suffix_pattern
server_url.regex[.case_sensitive]=regular_expression

server_url.address=ip_address|subnet|subnet_label

server_url.extension[.case_sensitive]=[.]filename_extension

server_url.host[.exact][.no_lookup]=host
server_url.host.[prefix|substring|suffix][.no_lookup]=string
server_url.host.regex[.no_lookup]=regular_expression
server_url.host.is_numeric=yes|no
server_url.host.no_name=yes|no

server_url.path[.case_sensitive]=/string
server_url.path[.substring|.suffix][.case_sensitive]=string
server_url.path.regex[.case_sensitive]=regular_expression

server_url.port={[low_port_number]..[high_port_number]|exact_port_number}

server_url.query.regex[.case_sensitive]=regular_expression

server_url.scheme=url_scheme
```

where all options are identical to `url=`, except for the URL being tested. For more information, see "url=" on page 137.

Discussion

The `server_url=` condition is identical to `url=`, except for the lack of a `define server_url` condition and `[server_url]` section. Most optimization in forwarding is done with `server_url.domain` conditions and sections.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, and `<Exception>` layers.

- Applies to all non-administrator transactions.

Examples

```
; Test if the server URL includes this pattern, and block access.
; Relative URLs, such as docs subdirectories and pages, will match.
server_url=http://www.example.com/docs access_server(no)

; Test if the URL host's IP address is a match.
server_url.address=10.1.198.0

; Test whether the URL includes company.com as domain.
server_url.domain=company.com

; Test whether the URL includes .com.
server_url.domain=.com

; Test if the URL includes this domain-suffix pattern,
; and block service. Relative URLs, such as docs
; subdirectories and pages, will match.
server_url.domain=company.com/docs access_server(no)

; examples of the use of server_url.extension=
server_url.extension=.txt
server_url.extension=(.htm, .html)
server_url.extension=(img, jpg, jpeg)

; This example matches the first request and doesn't match the second from
; the following two requests:
; http://1.2.3.4/test
; http://www.example.com
<forward>
server_url.host.is_numeric=yes

; In the example below we assume that 1.2.3.4 is the IP of the host mycompany
; The condition will match the following two requests if the reverse DNS was
; successful:
```

```
;request http://1.2.3.4/

;request http://mycompany.com/

; If the reverse DNS fails then the first request is not matched

<forward>

server_url.host.regex=mycompany


; server_url.path tests

; The following server_url.path strings would all match the example URL:

; http://www.example.com/cgi-bin/query.pl?q=test#fragment

server_url.path="/cgi-bin/query.pl?q=test"
server_url.path="/cgi-bin/query.pl"
server_url.path="/cgi-bin/"
server_url.path="/cgi" ; partial components match too

server_url.path="/" ; Always matches regardless of URL.


; testing the url port

server_url.port=80
```

See Also

- Conditions: `content_management=`, `url=`
- Definitions: `define subnet`, `define server_url.domain condition`

socks=

This condition is true whenever the session for the current transaction involves SOCKS to the client. The SOCKS=yes trigger is intended as a way to test whether or not a request arrived via the SOCKS proxy. It will be true for both SOCKS requests that the Proxy*SG* tunnels and for SOCKS requests the Proxy*SG* accelerates by handing them off to HTTP or IM. In particular, `socks=yes` remains true even in the resulting HTTP or IM transactions. Other triggers, such as `proxy.address` or `proxy.port` do not maintain a consistent value across the SOCKS transaction and the later HTTP or IM transaction, so they cannot be reliably used to do this kind of cross-protocol testing.

Replaces: `socks.`*`destination_address`*`=`

Syntax

```
socks=yes|no
```

Layer and Transaction Notes

- Use in all layers
- Applies to all proxy transactions.

See Also

- Conditions: `socks.accelerate=`
- Properties: `socks_gateway( )`, `socks.accelerate( )`, `socks.authenticate( )`, `socks.authenticate.force( )`.

socks.accelerated=

Tests whether the SOCKS proxy will hand off this transaction to other protocol agents for acceleration.

Syntax

```
socks.accelerated={yes|http|aol-im|msn-im|yahoo-im|no}
```

where:

- `yes` is true only for SOCKS transactions that will hand off to another protocol-specific proxy agent.
- `no` implies the transaction is a SOCKS tunnel.
- `http` is true if the transaction will be accelerated by the http proxy.
- `aol-im` is true if the transaction will be accelerated by the aol-im proxy.
- `msn-im` is true if the transaction will be accelerated by the msn-im proxy.
- `yahoo-im` is true if the transaction will be accelerated by the yahoo-im proxy.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Exception>` layers.
- Applies to SOCKS transactions.

See Also

- Conditions: `socks.method=`, `socks.version=`
- Properties: `socks_gateway( )`, `socks.accelerate( )`, `socks.authenticate( )`, `socks.authenticate.force( )`.

socks.method=

Tests the SOCKS protocol method name associated with the transaction.

Syntax

```
socks.method=CONNECT|BIND|UDP_ASSOCIATE
```

Layer and Transaction Notes

- Use in <Proxy> and <Exception> layers.
- Applies to SOCKS transactions.

See Also

- Conditions: ftp.method=, http.method=, http.x_method=, im.method=, method=, server_url=, socks.version=
- Properties: socks_gateway( ), socks.accelerate( ), socks.authenticate( ), socks.authenticate.force( ).

socks.version=

Tests whether the version of the SOCKS protocol used to communicate to the client is SOCKS 4/4a or SOCKS 5. SOCKS 5 has more security and is more highly recommended.

SOCKS 5 supports authentication and can be used to authenticate transactions that may be accelerated by other protocol services.

SOCKS 4/4a does not support authentication. If `socks.authenticate()` or `socks.authenticate.force()` is set during evaluation of a SOCKS 4/4a transaction, that transaction will be denied.

Syntax

```
socks.version=4..5
```

Layer and Transaction Notes

- Use in `<Proxy>`, `<Forward>`, and `<Exception>` layers.
- Applies to SOCKS transactions.
- Does not apply to administrator transactions.

Examples

This example authenticates SOCKS v5 clients, and allows only a known set of client IP addresses to use SOCKS v4/4a.

```
<Proxy>
    socks.version=5 socks.authenticate(my_realm )
    deny socks.version=4 client.address=!old_socks_allowed_subnet
```

See Also

- Conditions: `socks.destination_address=`, `socks.destination_port=`, `socks.method=`, `socks.version=`
- Properties: `socks_gateway( )`, `socks.accelerate( )`, `socks.authenticate( )`, `socks.authenticate.force( )`

streaming.client=

Tests the client agent associated with the current transaction.

Syntax

```
streaming.client=yes|no|windows_media|real_media|quicktime
```

where:

- `yes` is true if the user agent is recognized as a windows media player, real media player or quicktime player.
- `no` is true if the user agent is not recognized as a windows media player, real media player or quicktime player.
- other values are true if the user agent is recognized as a media player of the specified type.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, `<Forward>`, and `<Exception>` layers.
- Applies to HTTP and streaming transactions.
- Does not apply to administrator transactions.

See Also

- Conditions: `bitrate=`, `live=`, `streaming.content=`
- Properties: `access_server( )`, `max_bitrate( )`, `streaming.transport( )`

streaming.content=

Tests the content of the current transaction to determine whether or not it is streaming media, and to determine the streaming media type.

Syntax

```
streaming.content=yes|no|windows_media|real_media|quicktime
```

where:

- `yes` is true if the content is recognized as Windows media, Real media, or QuickTime content.
- `no` is true if the content is not recognized as Windows media, Real media, or QuickTime content.
- other values are true if the streaming content is recognized as the specified type.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, `<Forward>`, and `<Exception>` layers.
- Applies to all transactions.

See Also

- Conditions: `bitrate=`, `live=`, `streaming.client=`
- Properties: `access_server()`, `max_bitrate()`, `streaming.transport( )`

time=

Tests if the time of day is in the specified range or an exact match. The current time is determined by the ProxySG appliance's configured clock and time zone by default, although the UTC time zone can be specified by using the form `time.utc=`. The numeric pattern used to test the `time` condition can contain no whitespace.

Syntax

```
time[.utc]={[start_time]..[end_time]|exact_time}
```

where:

- *`start_time`*—Four digits (*nnnn*) in 24-hour time format representing the start of a time range; for example, `0900` specifies 9:00 a.m. If left blank, midnight (`0000`) is assumed.
- *`end_time`*—Four digits (*nnnn*) in 24-hour time format representing the end of a time range; for example, `1700` specifies 5:00 p.m. If left blank, `2359` (11:59 p.m.) is assumed.
- *`exact_time`*—Four digits (*nnnn*) in 24-hour time format representing an exact time.

Note: To test against an inverted range, such as a range that crosses from one day into the next, the following shorthand expression is available. While `time=(..0600|1900..)` specifies midnight to 6 a.m. and 7 p.m. to midnight, the policy language also recognizes `time=1900..0600` as equivalent.

Layer and Transaction Notes

- Use in all layers.
- Using time-related conditions to control caching behavior in a `<Cache>` layer may cause thrashing of the cached objects.
- Applies to all transactions.

Examples

```
; Tests for 3 a.m. to 1 p.m. UTC.
time.utc=0300..1300


; Allow access to a particular site only during 9 a.m.
; to noon UTC (presented in two forms).
; Restrict form:
<proxy>
   deny url.host=special_event.com time=!0900..1200


; Grant form:
<proxy>
   allow url.host=special_event.com time=0900..1200
```

```
; This example restricts the times during which certain
; stations can log in with administrative privileges.

define subnet restricted_stations
 10.10.10.4/30
 10.10.11.1
end subnet restricted_stations

<admin> client.address=restricted_stations

       allow time=0800..1800 weekday=1..5 admin.access=(READ||WRITE);
       deny
```

See Also

- Conditions: `date[.utc]=`, `day=`, `hour=`, `minute=`, `month=`, `weekday=`, `year=`

tunneled=

Tests if the current transaction represents a tunneled request. A tunneled request is one of:

- TCP tunneled request
- HTTP CONNECT request
- Unaccelerated SOCKS request

Note: HTTPS connections to the management console are not tunneled for the purposes of this test.

Syntax

```
tunneled=yes|no
```

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Applies to proxy transactions.

Examples

This example denies tunneled transactions except when they originate from the corporate subnet.

```
define subnet corporate_subnet
    10.1.2.0/24
    10.1.3.0/24
end

<Proxy>
   deny tunneled=yes client.address=!corporate_subnet
```

See Also

Conditions: `http.method=`, `socks.accelerated=`, `url.scheme=`

Properties: `sock.accelerate()`

url=

Tests if a portion of the requested URL matches the specified criteria. The basic `url=` test attempts to match the complete request URL against a specified pattern. The pattern may include the scheme, host, port, path and query components of the URL. If any of these is not included in the pattern, then the corresponding component of the request URL is not tested and can have any value.

Specific portions of the URL can be tested by applying URL component modifiers to the trigger. In addition to component modifiers, optional test type modifiers can be used to change the way the pattern is matched.

Note: This set of tests match against the originally requested URL, disregarding the effect of any `rewrite( )` actions. Because any rewrites of the URL intended for servers or other upstream devices must be respected by `<Forward>` layer policy, the `url=` triggers are not allowed in `<Forward>` layers. Instead, an equivalent set of `server_url=` tests are provided for use in the `<Forward>` layer. Those tests always take into account the effect of any `rewrite( )` actions on the URL.

Replaces: various `url_`*xxx* forms; `url.scheme=` replaces `protocol=`.

Syntax

```
url[.case_sensitive][.no_lookup]=prefix_pattern
url.domain[.case_sensitive][.no_lookup]=domain_suffix_pattern
url.regex[.case_sensitive]=regular_expression

url.address=ip_address|subnet|subnet_label

url.extension[.case_sensitive]=[.]filename_extension

url.host[.exact][.no_lookup]=host
url.host.[prefix|substring|suffix][.no_lookup]=string
url.host.regex[.no_lookup]=regular_expression
url.host.is_numeric=yes|no
url.host.no_name=yes|no

url.path[.case_sensitive]=/string
url.path[.substring|.suffix][.case_sensitive]=string
url.path.regex[.case_sensitive]=regular_expression

url.port={[low_port_number]..[high_port_number]|exact_port_number}

url.query.regex[.case_sensitive]=regular_expression

url.scheme=url_scheme
```

where the URL test patterns are:

- *prefix_pattern*—A URL pattern that includes at least a portion of the following:

  ```
  scheme://host:port/path
  ```

  Accepted prefix patterns include the following:

  ```
  scheme://host
  scheme://host:port
  scheme://host:port/path_query
  scheme://host/path_query
  //host
  ```

```
//host:port
//host:port/path_query
//host/path_query
host
host:port
host:port/path_query
host/path_query
/path_query
```

- *domain_suffix_pattern*—A URL pattern that includes a domain suffix, as a minimum, using the following syntax:

  ```
  scheme://domain_suffix:port/path
  ```

  Accepted domain suffix patterns include the following:

  ```
  scheme://domain_suffix
  scheme://domain_suffix:port
  scheme://domain_suffix:port/path_query
  scheme://domain_suffix/path_query
  //domain_suffix
  //domain_suffix:port
  //domain_suffix:port/path_query
  //domain_suffix/path_query
  domain_suffix
  domain_suffix:port
  domain_suffix:port/path_query
  domain_suffix/path_query
  ```

- *url_scheme*—One of `http`, `https`, `ftp`, `mms`, `rtsp`, `tcp`, `aol-im`, `msn-im`, or `yahoo-im`. The request URL has the scheme `https` only in the case of SSL termination. A request URL with the scheme `tcp` only has a host and a port, and occurs in two cases: when a connection is made to a TCP tunnel service port, and when the CONNECT method is used in an explicitly proxied HTTP request. For example, when the Web browser has an explicit HTTP proxy and the user requests an HTTPS URL, the browser creates a TCP tunnel using the CONNECT method.
- *host*—A domain name or IP address. Host names must be complete; for example, `url=http://www` fails to match a URL such as `http://www.example.com`. This use of a complete host instead of a *domain_suffix* (such as `example.com`) indicates the difference between the `url=` and `url.domain=` conditions.
- *domain_suffix*—A pattern which matches either a complete domain name or is a suffix of the domain name, respecting component boundaries. An IP address is not allowed. This use of a *domain_suffix* pattern instead of a complete host name marks the difference between the `url.domain=` and `url=` conditions.
- *port*—A port number, between 1 and 65535.
- *path_query*—The *path_query* portion of a URL is the string beginning with '`/`' that follows the host and port, and precedes any URL fragment. A *path_query* pattern is a string beginning with a '`/`' that matches the beginning of the *path_query*.
- *filename_extension*—A string representing a filename extension to be tested, optionally preceded by a period (`.`). A quoted empty string (`url.extension=""`) matches URLs that do not

include a filename extension, such as `http://example.com/` and `http://example.com/test`. To test multiple extensions, use parentheses and a comma separator (see the Example section below).

- *`regular_expression`*—A Perl regular expression. The expression must be quoted if it contains whitespace or any of the following: `& | ( ) < > { } ; ! . = " '`. For more information, refer to Appendix E: "Using Regular Expressions," in the Blue Coat Proxy*SG Configuration and Management Guide*.

Objects with paths relative to the *`prefix_pattern`* and *`domain_suffix_pattern`* are also considered a match (see the "Example" section).

The following are test modifiers:

- `.case_sensitive`—By default, all matching is case-insensitive; however, the matches on the path and query portions can be made case-sensitive by using the form `url.case_sensitive=`.
- *`.domain`*—Changes the way the match is performed on the host portion of the URL. The host pattern is a *`domain_suffix`* pattern which either matches the hostname exactly, or matches a suffix of the hostname on component boundaries. The host is converted to a domain name by reverse DNS lookup if necessary. For example, the condition `url.domain=//example.com` matches the request URL `http://www.example.com/`, but does not match the request URL `http://www.myexample.com/`.
- `.exact`—Forces an exact string comparison on the full URL or component.
- `.no_lookup`—Depending on the form of the request's host and the form of the pattern being matched, a DNS or reverse DNS lookup is performed to convert the request's host before the comparison is made. This lookup can be suppressed by using the `.no_lookup=` form of the condition. The `.no_lookup` modifier speeds up policy evaluation, but use of it may introduce loopholes into your security policy that can be exploited by those who want to bypass your security measures. DNS and reverse DNS lookups can be globally restricted by `restrict` definitions.
- `.prefix`—Test if the *`string`* pattern is a prefix of the URL or component.
- `.regex`—Test the URL or component against a *`regular_expression`* pattern.

  When applied to the `url=` condition, the URL is treated as a literal string for the purposes of the test.

  When applied to the `url.host=` condition, if the URL host was specified as an IP address, the behavior depends on whether or not the `no_lookup` modifier was specified. If `no_lookup` was specified, then the condition is false. If `no_lookup` was not specified, then a reverse DNS lookup is performed to convert the IP address to a domain name. If the reverse DNS lookup fails, then the condition is false. This leads to the following edge conditions: `url.host.regex=!""` has the same truth value as `url.host.no_name=yes`, and `url.host.regex.no_lookup=!""` has the same truth value as `url.host.is_numeric=yes`.

  When applied to the `url.host=` condition, this pattern match is always case-insensitive.
- `.substring`—Test if the *`string`* pattern is a substring of the URL or component. The substring need not match on a boundary (such as a subdomain or path directory) within a component.

- `.suffix`—Test if the *`string`* pattern is a suffix of the URL or component. The suffix need not match on a boundary (such as a domain component or path directory) within a URL component.

Note: `.prefix`, `.regex`, `.substring`, and `.suffix` are string comparisons that do not require a match on component boundaries. For this reason, `url.host.suffix=` differs from the host comparison used in `url.domain=` tests, which does require component level matches.

The URL component modifiers are:

- `.address`—Tests if the host IP address of the requested URL matches the specified IP address, IP subnet, or subnet definition. If necessary, a DNS lookup is performed on the host name. DNS lookups can be globally restricted by a `restrict DNS` definition.

  The patterns supported by the url.address= test are:

  - *`ip_address`*—Host IP address or subnet; for example, `10.1.198.0`.
  - *`subnet`*—A subnet mask; for example, `10.1.198.0/24`.
  - *`subnet_label`*—Label of a subnet definition block that binds a number of IP addresses or subnets.

  The `.address` modifier is primarily useful when the expression uses either a *`subnet`* or a *`subnet_label`*. If a literal *`ip_address`* is used, then the `url.address=` condition is equivalent to `url.host=`.

- `.host`—Tests the host component of the requested URL against the IP address or domain name specified by the *`host`* pattern. The pattern cannot include a forward slash (`/`) or colon (`:`). It does not recognize wild cards or suffix matching. Matches are case-insensitive. The default test type is `.exact`.

Note: `url.host.exact=` can be tested using hash techniques rather than string matches, and will therefore have significantly better performance than other, string based, versions of the `url.host=` tests. .

  Since the host component of a request URL can be either an IP address or a domain name, a conversion is sometimes necessary to allow a comparison.

  - If the expression uses a domain name and the host component of the request URL is an IP address, then the IP address is converted to a domain name by doing a reverse DNS lookup.
  - If the expression uses an IP address and the host component of the request URL is a domain name, then the domain name is converted to an IP address by doing a DNS lookup.

  The `.host` component supports additional test modifiers:

  - `.is_numeric`—This is true if the URL host was specified as an IP address. For some types of transactions (for example, transparent requests on a non-accelerated port), this condition will always be true.
  - `.no_name`—This is true if no domain name can be found for the URL host. Specifically, it is true if the URL host was specified as an IP address, and a reverse DNS lookup on this IP address fails, either because it returns no name or a network error occurs.

- `.path`—Tests the path component of the request URL. By default, the pattern is tested as a prefix of the complete path component of the requested URL, as well as any query component. The path and query components of a URL consist of all text from the first forward slash (`/`) that follows the host or port, to the end of the URL, not including any fragment identifier. The leading forward

slash is always present in the request URL being tested, because the URL is normalized before any comparison is performed. Unless an `.exact`, `.substring`, or `.regex` modifier is used, the pattern specified must include the leading '/' character.

In the following URL example, bolding shows the components used in the comparison; `?q=test` is the included query component and `#fragment` is the ignored fragment identifier:

```
http://www.example.com/cgi-bin/query.pl?q=test#fragment
```

A URL such as the following is normalized so that a forward slash replaces the missing path component: `http://www.example.com` becomes `http://www.example.com/`.

- `.port`—Tests if the port number of the requested URL is within the specified range or an exact match. URLs that do not explicitly specify a port number have a port number that is implied by the URL scheme. The default port number is 80 for HTTP, so `url.port=80` is true for any HTTP-based URL that does not specify a port.

  The patterns supported by the `url.address=` test are:

  - *`low_port_number`*—A port number at the low end of the range to be tested. Can be a number between 1 and 65535.
  - *`high_port_number`*—A port number at the high end of the range to be tested. Can be a number between 1 and 65535.
  - *`exact_port_number`*—A single port number; for example, `80`. Can be a number between 1 and 65535.

  Note that the numeric pattern used to test the `url.port` condition can contain no whitespace.

- `.query`—Tests if the regex matches a substring of the query string component of the request URL. If no query string is present, the test is false. As a special case, `url.query_regex=!""` is true if there is no query string.

  The query string component of the request URL, if present, consists of all text from the first '?' following the path, to the end of the URL, or up to the first occurrence of '#', whichever comes first. Thus, any fragment identifier that might be present is excluded from the query string component. If there is a query string component at all, then it begins with a '?' character.

- `.scheme`—Tests if the scheme of the requested URL matches the specified schema string. The comparison is always case-insensitive.

Discussion

The `url=` condition can be considered a convenient way to do testing that would require a combination of the following conditions: `url.scheme=`, `url.host=`, `url.port=`, and `url.path=`. For example,

```
url=http://example.com:8080/index.html
```

is equivalent to:

```
url.scheme=http url.host=example.com url.port=8080 url.path=/index.html
```

If you are testing a large number of URLs using the `url=` condition, consider the performance benefits of a `url` definition block or a `[url]` section (see Chapter 6: "Definition Reference").

If you are testing a large number of URLs using the `url.domain=` condition, consider the performance benefits of a `url.domain` definition block or a `[url.domain]` section (see Chapter 6: "Definition Reference").

Regular expression matches are not anchored. You may want to use either or both of the ^ and $ operators to anchor the match. Alternately, use the `.exact`, `.prefix`, or `.suffix` form of the test, as appropriate.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, and `<Exception>` layers.
- Applies to all non-administrator transactions.

Examples

```
; Test if the URL includes this pattern, and block service.
; Relative URLs, such as docs subdirectories and pages, will match.
url=http://www.example.com/docs service(no)

; Test if the URL host's IP address is a match.
url.address=10.1.198.0

; Test whether the URL includes company.com as domain.
url.domain=company.com

; Test whether the URL includes .com.
url.domain=.com

; Test if the URL includes this domain-suffix pattern,
; and block service. Relative URLs, such as docs
; subdirectories and pages, will match.
url.domain=company.com/docs service(no)

; examples of the use of url.extension=
url.extension=.txt
url.extension=(.htm, .html)
url.extension=(img, jpg, jpeg)

; This example matches the first request and doesn't match the second from
; the following two requests:
; http://1.2.3.4/test
```

```
; http://www.example.com
<proxy>
url.host.is_numeric=yes;


; In the example below we assume that 1.2.3.4 is the IP of the host mycompany
; The condition will match the following two requests if the reverse DNS was
; successful:
;request http://1.2.3.4/
;request http://mycompany.com/
; If the reverse DNS fails then the first request is not matched
<proxy>
url.host.regex=mycompany


; url.path tests
; The following server_url.path strings would all match the example URL:
; http://www.example.com/cgi-bin/query.pl?q=test#fragment
url.path="/cgi-bin/query.pl?q=test"
url.path="/cgi-bin/query.pl"
url.path="/cgi-bin/"
url.path="/cgi" ; partial components match too
url.path="/" ; Always matches regardless of URL.


; testing the url port
url.port=80
```

See Also

- Conditions: `category=`, `console_access=`, `content_management=`, `server_url=`
- Definitions: `define subnet`, `define url condition`, `define url.domain condition`

user=

Tests the authenticated username associated with the transaction. This trigger is only available if the transaction was authenticated (that is, the `authenticate( )` property was set to something other than `no`, and the `proxy_authentication( )` property was not set to `no`).

Syntax

```
user=user_name
```

where *user_name* is a username.

- NTLM realm: Usernames are case-insensitive.

  In NTLM this provides the flexibility of matching either a full username (which includes the NT Domain) or relative username (which does not include the NT Domain).

  For example:

  ```
  user=bluecoat\mary.jones
  ```

  matches a complete username, and

  ```
  user=mary.jones
  ```

  matches a relative name.

- UNIX (local) realm: Usernames are case-sensitive.
- RADIUS realm: Username case-sensitivity depends on the RADIUS server's setting. The case-sensitive setting should also be set correctly when defining a RADIUS realm in the Proxy*SG*.
- LDAP realm: Username case-sensitivity depends on the LDAP server's setting. The case-sensitive setting should also be set correctly when defining an LDAP realm in Proxy*SG*.

  In LDAP this provides the flexibility of matching either a fully qualified domain name or relative username.

  For example:

  ```
  user="cn=mary.jones,cn=sales,dc=bluecoat,dc=com"
  ```

  or

  ```
  user="uid=mary.jones,ou=sales,o=bluecoat"
  ```

  matches a complete username, and

  ```
  user=mary.jones
  ```

  matches a relative name.

Layer and Transaction Notes

- Use in `<Admin>` and `<Proxy>` layers.

Examples

```
; Test for user john.smith.
user=john.smith
```

See Also

- Conditions: `attribute.`*`name`*`=`, `authenticated=`, `group=`, `has_attribute.`*`name`*`=`, `http.transparent_authentication=`, `realm=`, `user.domain=`
- Properties: `authenticate( )`, `authenticate.force( )`, `check_authorization( )`, `deny.unauthorized( )`, `socks.authenticate( )`, `socks.authenticate.force( )`

user.domain=

Tests if the client is authenticated, the logged-into realm is an NTLM realm, and the domain component of the username is the specified domain. If all of these conditions are met, the response will be true. This trigger is unavailable if the current transaction is not authenticated (that is, the `authenticate( )` property is set to `no`).

Replaces: `user_domain=`

Syntax

```
user.domain=windows_domain_name
```

where *windows_domain_name* is a Windows domain name. This name is case-insensitive.

Layer and Transaction Notes

- Use in `<Admin>` and `<Proxy>` layers.

Examples

```
; Test if the user is in domain all-staff.
user.domain=all-staff
```

See Also

- Conditions: `attribute.name=`, `authenticated=`, `group=`, `has_attribute.name=`, `http.transparent_authentication=`, `realm=`, `user=`
- Properties: `authenticate( )`, `authenticate.force( )`, `check_authorization( )`, `deny.unauthorized( )`, `socks.authenticate( )`, `socks.authenticate.force( )`

user.x509.issuer=

Tests the issuer of the x509 certificate used in authentication to certificate realms. The `user.x509.issuer=` condition is primarily useful in constructing explicit certificate revocation lists. This condition will only be true for users authenticated against a certificate realm.

Syntax

```
user.x509.issuer=issuer_DN
```

where *`issuer_DN`* is an RFC2253 LDAP DN, appropriately escaped. Comparisons are case-sensitive.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Admin>`, and `<Exception>` Layers.
- Applies to proxy transactions.

See Also

- Conditions: `user.x509.serialNumber=`, `user.x509.subject=`
- Properties: `authenticate( )`, `authenticate.force( )`

user.x509.serialNumber=

Tests the serial number of the x509 certificate used to authenticate the user against a certificate realm. The `user.x509.serialNumber=` condition is primarily useful in constructing explicit certificate revocation lists. Comparisons are case-insensitive.

Syntax

```
user.x509.serialNumber=serial_number
```

where *`serial_number`* is a string representation of the certificate's serial number in HEX.

The string is always an even number of characters long, so if the number needs an odd number of characters to represent in hex, there is a leading zero. This can be up to 160 bits.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Admin>`, and `<Exception>` Layers.
- Applies to proxy transactions.

See Also

- Conditions: `user.x509.issuer=`, `user.x509.subject=`
- Properties: `authenticate( )`, `authenticate.force( )`

user.x509.subject=

Tests the subject field of the x509 certificate used to authenticate the user against a certificate realm. The `user.x509.subject=` condition is primarily useful in constructing explicit certificate revocation lists.

Syntax

```
user.x509.subject=subject
```

where *`subject`* is an RFC2253 LDAP DN, appropriately escaped.

Comparisons are case-sensitive.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Admin>`, and `<Exception>` Layers.
- Applies to proxy transactions.

See Also

- Conditions: `user.x509.issuer=`, `user.x509.serialNumber=`
- Properties: `authenticate( )`, `authenticate.force( )`

weekday=

Tests if the day of the week is in the specified range or an exact match. By default, the ProxySG appliance's date is used to determine the day of the week. To specify the UTC time zone, use the form `weekday.utc=`. The numeric pattern used to test the `weekday=` condition can contain no whitespace

Syntax

```
weekday[.utc]={[first_weekday]..[last_weekday]|exact_weekday}
```

where:

- *`first_weekday`*—An integer from 1 to 7, where 1 specifies Monday and 7 specifies Sunday, indicating the first day of the week that tests true. If left blank, Monday is assumed.
- *`last_weekday`*—An integer from 1 to 7, where 1 specifies Monday and 7 specifies Sunday, indicating the last day of the week that tests true. If left blank, Sunday is assumed.
- *`exact_weekday`*—An integer from 1 to 7, where 1 specifies Monday and 7 specifies Sunday, indicating the day of the week that tests true.

Note: When you want to test a range that wraps from one week into the next, the following shorthand expression is available. While `weekday=(..1|6..)` specifies a long weekend that includes Monday, the policy language also recognizes `weekday=6..1` as equivalent.

Layer and Transaction Notes

- Use in all layers.
- Using time-related conditions to control caching behavior in a `<Cache>` layer may cause thrashing of the cached objects.
- Applies to all transactions.

Examples

```
; Test for the weekend.
weekday=6..7


; Test for Saturday through Monday.
weekday=6..1
```

See Also

- Conditions: `date[.utc]=`, `day=`, `hour=`, `minute=`, `month=`, `time=`, `year=`

year=

Tests if the year is in the specified range or an exact match. The current year is determined by the date set on the Proxy*SG* by default. To specify the UTC time zone, use the form `year.utc=`. Note that the numeric pattern used to test the `year=` condition can contain no whitespace.

Syntax

```
year[.utc]={[first_year]..[last_year]|exact_year}
```

where:

- *`first_year`*—Four digits (*nnnn*) representing the start of a range of years; for example, `2002`.
- *`last_year`*—Four digits (*nnnn*) representing the end of a range of years. If left blank, all years from first_year on are assumed.
- *`exact_year`*—Four digits (*nnnn*) representing an exact year.

Note: To test against an inverted range of years, the following shorthand expression is available. While `year=(..1998|2003..)` specifies years up to and including 1998, and from 2003 on, the policy language also recognizes `year=2003..1998` as equivalent.

Layer and Transaction Notes

- Use in all layers.
- Using time-related conditions to control caching behavior in a `<Cache>` layer may cause thrashing of the cached objects.
- Applies to all transactions.

Examples

```
; Tests for the years 2003 through 2005.
year=2003..2005
```

See Also

- Conditions: `date[.utc]=`, `day=`, `hour=`, `minute=`, `month=`, `time=`, `weekday=`, `year=`

Chapter 4: *Property Reference*

A *property* is a variable that can be set to a value. At the beginning of a transaction, all properties are set to their default values. As each layer in the policy is evaluated in sequence, it can set a property to a particular value. A property retains the final value setting when evaluation ends, and the transaction is processed accordingly. Properties that are not set within the policy maintain their default values.

Property Reference

The remainder of this chapter lists the properties and their accepted values. It also provides tips as to where each property can be used and examples of how to use them.

access_log( )

Selects the access log used for this transaction. Multiple access logs can be selected to record a single transaction. Individual access logs are referenced by the name given in configuration. Configuration also determines the format of the each log. For more information on logging, refer to Chapter 19: "Access Logging," in the ProxySG *Configuration and Management Guide*.

To record entries in the event log, see "log_message( )" on page 232.

Syntax

```
access_log(auto|no|log_name_list)
access_log.log_name(yes|no)
access_log.[log_name_list](yes|no)
```

The default value is `auto`.

where:

- `auto`—use the default log for this protocol.
- `no`—turns off logging, either for this transaction or to the specified *`log_name`* or *`log_name_list`*.
- `yes`—turns on logging for this transaction to the specified *`log_name`* or *`log_name_list`*.
- *`log_name`*—an access log name as defined in configuration
- *`log_name_list`*—a list of access log names as defined in configuration, of the form:

  ```
  log_name_1, log_name_2, ...
  ```

Discussion

Each of the syntax variants has a different role in selecting the list of access logs used to record the transaction:

- `access_log( )` overrides any previous access log selections for this transaction.
- `access_log.log_name( )` selects or de-selects the named log, according to the specified value. Any other log selections for the transaction are unaltered.
- `access_log.[log_name_list]( )` selects or de-selects all the logs named in the list, according to the specified value. The selection of logs not named in the list is unaffected.

Layer and Transaction Notes

- Use in all but `<Admin>` layers.
- Applies to proxy transactions.

See Also

- Properties: `log.suppress.`*`field-id`*, `log.rewrite.`*`field-id`*`( )`
- Actions: `log_message( )`

access_server( )

Determines whether the client can receive streaming content directly from the origin content server or other upstream device. Set to `no` to serve only cached content.

Note: Since part of a stream can be cached, and another part of the same stream can be uncached, `access_server(no)` can cause a streaming transaction to be terminated after some of the content has been served from the cache.

Syntax

```
access_server(yes|no)
```

The default value is `yes`.

Layer and Transaction Notes

- Use in `<Forward>` layers to replace `allow | deny( )`. The `access_server(no)` property is equivalent to `deny( )` for a `<Forward>` layer.
- Use in `<Proxy>`, `<Cache>`, and `<Forward>` layers.
- Applies to HTTP, SOCKS, and streaming transactions.

See Also

- Conditions: `bitrate=`, `live=`, `streaming.client=`, `streaming.content=`

action( )

Selectively enables or disables a specified define action block. The default value is no.

Note: Several define action blocks may be enabled for a transaction. If more than one action selected rewrites the URL or header a specific header, the actions are deemed to conflict and only one will be executed. When detected at runtime, action conflicts will be reported in the event log as a severe event. Action conflicts may also be reported at compilation time.

Replaces: `action(action_label)` replaces `label(action_label)`

Syntax

```
action(action_label)
action.action_label(yes|no)
```

The default value is `no` for all defined actions.

where *`action_label`* is the label of the `define action` block to be enabled or disabled.

Discussion

Each of the different syntax variants has a different role in selecting the list of actions applied to the transaction:

- `action()` enables the specified action block and disables all other actions blocks.
- *`action.action_label( )`* enables or disables the specific action block. Any other action block selections for the transaction are unaltered.

Layer and Transaction Notes

- Use in `<Cache>`, `<Proxy>`, and `<Exception>` layers. The actions specified in the action block must be appropriate to the referencing layer.

See Also

- Definitions: `define action`

advertisement( )

Determines whether to treat the objects at a particular URL as banner ads to improve performance. If the content is not specific to a particular user or client, then the hit count on the origin server is maintained while the response time is optimized using the following behavior:

- Always serve from the cache if a cached response is available. Ignore any request headers that bypass the cache; for example, `Pragma: No-Cache`.
- Always cache the response from the origin server, similar to `force_cache(all)`.
- If the request was served from the cache, request the object from the origin server in the background to maintain the origin server's hit count on the ad and also allow ad services to deliver changing ads.

A number of CPL properties affect caching behavior, as listed in the "See Also" section below. Remember that any conflict between their settings is resolved by CPL's evaluation logic, which uses the property value that was last set when evaluation ends.

Syntax

```
advertisement (yes|no)
```

The default value is `no`.

Layer and Transaction Notes

- Use in `<Cache>` layers.
- Do not use in `<Proxy>` layers.
- Applies to HTTP transactions, except FTP over HTTP transactions.

See Also

- Properties: `always_verify( )`, `cache( )`, `cookie_sensitive( )`, `pipeline( )`, `refresh( )`, `ttl( )`, `ua_sensitive( )`

allow

Allows the transaction to be served.

Allow can be overridden by the `access_server( )`, `deny( )`, `force_deny( )`, `authenticate( )`, `exception( )`, or `force_exception( )` properties or by the `redirect( )` action.

Allow overrides `deny( )` and `exception( )` properties.

Note: Caution should be exercised when using `allow` in layers evaluated after layers containing `deny`, to ensure that security is not compromised.

Replaces: `service(yes)`

Syntax

```
allow
```

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, and `<Admin>` layers.
- Do not use in `<Forward>` layers. Use "access_server( )" on page 155.
- Applies to all transactions.

See Also

- Properties: `access_server( )`, `deny( )`, `force_deny( )`, `authenticate( )`, `exception( )`, `force_exception( )`
- Actions: `redirect( )`

always_verify( )

Determines whether each request for the objects at a particular URL must be verified with the origin server. This property provides a URL-specific alternative to the global `caching` setting `always-verify-source`. If there are multiple simultaneous accesses of an object, the requests are reduced to a single request to the origin server.

Syntax

```
always_verify(yes|no)
```

The default value is `no`.

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Applies to HTTP proxy transactions, except FTP over HTTP transactions.

See Also

- Properties: `advertisement( )`, `bypass_cache( )`, `cache( )`, `cookie_sensitive( )`, `force_cache( )`, `pipeline( )`, `refresh( )`, `ttl( )`, `ua_sensitive( )`

authenticate( )

Identifies the realm used to authenticate the user associated with the current transaction. Authentication realms are referenced by the name given in configuration.

If the transaction has already been authenticated in the same realm by the SOCKS proxy, no new authentication challenge is issued. If the realms identified in the `socks.authenticate( )` and `authenticate( )` actions differ however, a new challenge is issued.

How authentication is performed is a function of the capabilities of the realm, the protocol involved, and the setting of the `authenticate.mode( )` property.

The `authenticate( )` action has higher precedence than `allow`, so a subsequent allow does not prevent an authentication challenge.

The relation between authentication and denial is controlled through the `authenticate.force( )` property. The default setting `no` implies that denial overrides `authenticate( )`, with the result that user names may not appear for denied requests if that denial could be determined without authentication. To ensure that user names appear in access logs, use `authenticate.force(yes)`.

Syntax

```
authenticate(no)
authenticate(realm_name[, display_name])
```

The default value is `no`.

where:

- `no`—User authentication is not required for this transaction. No authentication challenge is issued.
- *`realm_name`*—A realm that must be authenticated against. An authentication challenge may be issued.
- *`display_name`*—A string that is displayed in the Web browser when credentials are requested in place of *`realm_name`*.

Discussion

The `authenticate( )` property may result in the following exceptions, testable with the `exception.id=` trigger in an `<Exception>` layer.

- `authentication_failed`—The offered credentials were not valid in this authentication realm.
- `authentication_failed_password_expired`—Authentication failed due to password expiry.
- `configuration_error`—Authentication failed due to a realm configuration error.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Admin>` layers.
- Applies to proxy and administrator transactions.

Example

```
; Require authentication for internet access.
<proxy>
```

```
url.domain = !corporate.com authenticate(OurRealm, “log in for internet access”)
```

The next example illustrates the relation between authentication and denial. All users outside an allowed subnet are denied before authentication. They are not allowed to submit credentials to the authentication server. Users within the allowed subnet are authenticated regardless of whether they will eventually be allowed or denied, so their user names are available for the access log.

```
define allowed_source_ip
    10.1.2.0/24 ; my subnet(s)
    ;...
end

<proxy>

   authenticate( myrealm )

<proxy>

   deny client.address=!allowed_source_ip ; denied before authentication
   authenticate.force(yes)                ; all others denied after

<proxy>

   deny category=(Sports, Gambling) ; would deny before auth except for force.
```

See Also

- Conditions: `authenticated=`, `exception.id=`, `group=`, `has_attribute.name=`, `http.transparent_authentication=`, `realm=`, `user=`, `user_domain=`
- Properties: `authenticate.force( )`, `authenticate.mode( )`, `authenticate.use_url_cookie( )`, `check_authorization( )`, `socks.authenticate( )`, `socks.authenticate.force( )`

authenticate.force( )

This property controls the relation between authentication and denial.

Syntax

```
authenticate.force(yes|no)
```

The default value is `no`.

where:

- `yes` —Makes an `authenticate( )` higher priority than `deny( )`or `exception( )`. Use `yes` to ensure that userID's are available for access logging (including denied requests).
- `no`—`deny( )` and `exception( )` have a higher priority than `authenticate( )`. This setting allows early denial.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Admin>` layers and transactions.
- Does not apply to `<Cache>` layers or transactions.

See Also

- Conditions: `authenticated=`, `group=`, `has_attribute.name=`, `http.transparent_authentication=`, `realm=`, `user=`, `user_domain=`
- Properties: `authenticate( )`, `check_authorization( )`, `socks.authenticate( )`, `socks.authenticate.force( )`

authenticate.mode( )

Using the `authentication.mode( )` property selects a combination of challenge type and surrogate credentials.

Challenge type is what kind of challenge (proxy, origin or origin-redirect) is issued.

Surrogate credentials are credentials accepted in place of the user's real credentials. They are used for a variety of reasons. Blue Coat supports three kinds of surrogate credentials.

- *IP* surrogate credentials authenticate the user based on the IP address of the client. Once any client has been successfully authenticated, all future requests from that IP address are assumed to be from the same user.
- *Cookie* surrogate credentials use a cookie constructed by the Proxy*SG* as a surrogate. The cookie contains information about the user, so multiple users from the same IP address can be distinguished. The cookie contains a temporary password to authenticate the cookie; this password expires when the credential cache entry expires.
- *Connection* surrogate credentials use the TCP/IP connection to authenticate the user. Once authentication is successful, the connection is marked authenticated and all future requests on that connection are considered to be from the same user.

In SGOS 3.1.x, the connection's authentication information includes the realm in which it was authenticated. The surrogate credentials are accepted only if the current transaction's realm matches the realm in which the session was authenticated.

Syntax

```
authenticate.mode(mode_type)
```

where *mode_type* is one of the following, shown followed by the implied challenge type and surrogate credential:

- `auto`—Allows the Proxy*SG* to make a best effort to determine a suitable authentication mechanism for the transaction. For streaming transactions, `authenticate.mode(auto)` uses origin mode.
- `legacy`—The default for systems upgraded from SGOS 2.x.
- `proxy` (proxy/connection)—Specifies a *normal* forward proxy. In some situations proxy challenges will not work; *origin* challenges are then issued.
- `proxy-ip` (proxy/IP)—Specifies an insecure forward proxy, possibly suitable for LANs of single-user workstations. Mode switching occurs as for proxy.
- `origin` (origin/connection)—Acts as a normal *Web server*. In this case, no forwarding of credentials is needed.
- `origin-ip` (origin/IP)–Used to support NTLM authentication to the upstream device, and the client cannot handle cookie credentials. This mode is primarily used for automatic downgrading, but it can be selected for specific situations.

  This mode is insecure: after a user has authenticated from an IP address, all further requests from that IP address are treated as from that user. If the client is behind a NAT, or on a multi-user system, this can present a serious security problem.

- `origin-cookie` (origin/cookie)—Used in forward proxies to support pass-through authentication more securely than `origin-ip` if the client understands cookies. Only the HTTP and HTTPS protocols support cookies; other protocols are automatically downgraded to `origin-ip`.

  This mode could also be used in reverse proxy situations if impersonation is not possible and the origin server requires authentication.

- `origin-cookie-redirect` (origin-redirect/cookie)—The SGOS 2.x transparent cookie mode, it is intended to be used in forward proxies. The Proxy*SG* authenticates the user to a separate virtual host. BASIC and NTLM authentication can be forwarded, but the client must support both redirects and cookies.

The default value is `auto`.

Layer and Transaction Notes

- Use in `<Proxy>` layers
- Applies to proxy transactions.

authenticate.use_url_cookie( )

This property is used to authenticate users who have third party cookies explicitly disabled.

Note: With a value of `yes`, if there is a problem loading the page (you get an error page or you cancel an authentication challenge), the `cfauth` cookie is displayed. You can also see the cookie in packet traces, but not in the browser URL window or history under normal operation.

Syntax

```
authenticate.use_url_cookie(yes|no)
```

The default is `no`.

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Applies to HTTP proxy transactions.

See Also

Properties: `authenticate.mode( )`

block_category( )

This property has been deprecated.

In current CPL, the use of `block_category(`*`category_list`*`)` has been replaced by

```
category=category_list exception(content_filter_denied)
```

However, `block_category()` will be overridden by `content_filter_override(yes)`, while this is not the case for the replacement CPL code shown above. Note that `content_filter_override()` is also deprecated.

bypass_cache( )

Determines whether the cache is bypassed for a request. If set to `yes`, the cache is not queried and the response is not stored in the cache. Set to `no` to specify the default behavior, which is to follow standard caching behavior.

While static and dynamic bypass lists allow traffic to bypass the cache based on the destination IP address, the `bypass_cache` property is intended to allow a bypass based on the properties of the client; for example, you might use it to allow specific users or user groups to bypass the cache.

This property has no effect on streaming objects.

Syntax

```
bypass_cache(yes|no)
```

The default is `no`.

Layer and Transaction Notes

- Use only in `<Proxy>` layers.
- Applies to HTTP, HTTPS, FTP over HTTP, and transparent FTP transactions.

Example

```
; Bypass the cache for requests from this client IP address.
client.address=10.25.198.0 bypass_cache(yes)
```

See Also

- Properties: `advertisement( )`, `always_verify( )`, `cache( )`, `cookie_sensitive( )`, `direct( )`, `dynamic_bypass`, `force_cache( )`, `pipeline( )`, `refresh( )`, `ttl( )`, `ua_sensitive( )`

cache( )

Controls HTTP and FTP caching behavior. A number of CPL properties affect caching behavior.

- If `bypass_cache(yes)` is set, then the cache is not accessed and the value of `cache( )` is irrelevant.
- If `cache(yes)` is set, then the `force_cache(all)` property setting modifies the definition of what is considered a cacheable response.
- The properties `cookie_sensitive(yes)` and `ua_sensitive(yes)` have the same effect on caching as `cache(no)`.

Other CPL properties that affect caching behavior are listed in the "See Also" section below. Remember that any conflict between their settings is resolved by CPL's evaluation logic, which uses the property value that was last set when evaluation ends.

Syntax

```
cache(yes|no)
```

The default is `yes`.

where:

- `yes`—Specifies the default behavior: cache responses from the origin server if they are cacheable.
- `no`—Do not store the response in the cache, and delete any object that was previously cached for this URL.

Layer and Transaction Notes

- Use only in `<Cache>` layers.
- Applies to proxy transactions.

Example

```
; Prevent objects at this URL from being added to the cache.
url=http://www.example.com/docs cache(no)


; This example shows use of cache(yes) in an exception to broader no-cache policy.
define url.domain condition non_cached_sites
   http://example1.com
   http://example2.com
end
<cache>
   condition=non_cached_sites cache(no)
<cache>
   url.extension=(gif, jpg) cache(yes) ; OK to cache these filetypes regardless.
```

See Also

- Properties: `advertisement( )`, `always_verify( )`, `bypass_cache( )`, `cookie_sensitive( )`, `direct( )`, `dynamic_bypass`, `force_cache()`, `pipeline( )`, `refresh( )`, `ttl( )`, `ua_sensitive( )`

check_authorization( )

In connection with CAD (Caching Authenticated Data) and CPAD (Caching Proxy-Authenticated Data) support, `check_authorization( )` is used when you know that the upstream device sometimes (not always or never) requires the user to authenticate and be authorized for this object.

Setting the value to `yes` results in a GIMS (Get If Modified Since) to check authorization upstream, and the addition of a `"Cache-Control: must-revalidate"` header to the downstream response.

Syntax

```
check_authorization(yes|no)
```

The default is `no`.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Cache>` layers.
- Applies to HTTP and RTSP proxy transactions.

See Also

- Conditions: `authenticated=`, `group=`, `has_attribute.name=`, `http.transparent_authentication=`, `realm=`, `user=`, `user_domain=`
- Properties: `authenticate( )`, `authenticate.force( )`

content_filter_override( )

This property has been deprecated.

`content_filter_override(yes)` has two effects:

- It prevents the request from being sent to the off-box content filter, if off-box content filtering is configured. In this case, it is equivalent to `request.filter_service(no).`
- It suppresses denial of service based on on-box content filter categories specified using `block_category()`, another deprecated command. However, it has no effect on denial of service specified by CPL rules using the `category=` condition combined with `exception()` or `deny`.

The default value is `no`.

If you use `content_filter_override(yes)` to disable off-box content filtering, switch to `request.filter_service(no)` instead.

However, if you use `content_filter_override(yes)` to disable on-box content filtering that is specified using block_category(...), rewrite your policy to replace `content_filter_override()` and `block_category()` with `category=`, `exception(content_filter_denied)`, and `allow`.

For more information, see "request.filter_service( )" on page 210.

cookie_sensitive( )

Used to modify caching behavior by declaring that the object served by the request varies based on cookie values. Set to `yes` to specify this behavior, or set to `no` for the default behavior, which caches based on HTTP headers.

Using `cookie_sensitive(yes)` has the same effect as `cache(no)`.

There are a number of CPL properties that affect caching behavior, as listed in the "See Also" section below. Remember that any conflict between their settings is resolved by CPL's evaluation logic, which uses the property value that was last set when evaluation ends.

Syntax

```
cookie_sensitive(yes|no)
```

The default value is `no`.

Layer and Transaction Notes

- Use in `<Cache>` layers.
- Applies to HTTP proxy transactions, except FTP over HTTP transactions.

See Also

- Properties: `advertisement( )`, `always_verify( )`, `bypass_cache( )`, `cache( )`, `direct( )`, `force_cache( )`, `pipeline( )`, `refresh( )`, `ttl( )`, `ua_sensitive( )`

delete_on_abandonment( )

If set to `yes`, specifies that if all clients who may be simultaneously requesting a particular object close their connections before the object is delivered, the object fetch from the origin server is abandoned, and any prior instance of the object is deleted from the cache.

Syntax

```
delete_on_abandonment(yes|no)
```

The default value is `no`.

Layer and Transaction Notes

- Use in `<Cache>` layers.
- Applies to proxy transactions.

See Also

- Properties: `advertisement( )`, `always_verify( )`, `bypass_cache( )`, `cache( )`, `cookie_sensitive( )`, `direct( )`, `dynamic_bypass( )`, `force_cache()`, `pipeline( )`, `refresh( )`, `ttl( )`, `ua_sensitive( )`

deny( )

Denies service.

Denial can be overridden by `allow` or `exception( )`. To deny service in a way that cannot be overridden by a subsequent `allow`, use `force_deny( )` or `force_exception( )`.

The relation between `authenticate( )` and `deny( )` is controlled by the `authenticate.force( )` property. By default, `deny( )` overrides `authenticate( )`. Recall that this means that a transaction can be denied before authentication occurs, resulting in no user indentification available for logging.

Similarly, the relation between `socks.authenticate( )` and `deny( )` is controlled by the `socks.authenticate.force( )` property. By default, `deny( )` overrides `socks.authenticate( )`.

Replaces: `service(no)`

Syntax

```
deny
deny(details)
```

where *details* is a string defining a message to be displayed to the user. The details string may contain CPL substitution variables.

Discussion

The `deny(details)` property is equivalent to `exception(policy_denied, details)`. The identity of an exception being returned can be tested in an `<Exception>` layer using `exception.id=`.

For HTTP, a *policy_denied* exception results in a 403 Forbidden response. This is appropriate when the denial does not depend on the user identity. When the denial does depend on user identity, use `deny.unauthorized( )` instead to give the user an opportunity to retry the request with different credentials.

Layer and Transaction Notes

- Use in `<Cache>`, `<Proxy>`, and `<Admin>` layers. In `<Forward>` layers, use "access_server( )" on page 155.
- Applies to all transactions.

Example

```
deny url.address=10.25.100.100
```

See Also

- Condition: `exception.id=`
- Properties: `allow`, `authenticate.force( )`, `deny.unauthorized( )`, `force_deny( )`, `never_refresh_before_expiry( )`, `never_serve_after_expiry( )`, `remove_IMS_from_GET( )`, `remove_PNC_from_GET( )`, `remove_reload_from_IE_GET( )`, `request.filter_service( )`, `socks.authenticate( )`, `socks.authenticate.force( )`

deny.unauthorized( )

The `deny.unauthorized` property instructs the Proxy*SG* to issue a challenge (401 Unauthorized or 407 Proxy authorization required). This indicates to the client that the resource cannot be accessed with their current identity, but might be accessible using a different identity. The browsers typically respond by bringing up a dialog box so the user can change their identity. (The *`details`* string appears in the challenge page so that if the user cancels, there is some additional help information provided).

Typically, use `deny( )` if the policy rule forbids everyone access, but use `deny.unauthorized` if the policy rule forbids only certain people.

Syntax

```
deny.unauthorized
deny.unauthorized(details)
```

where *`details`* is a string defining a message to be displayed to the user. The details string may contain CPL substitution variables.

Discussion

If current policy contains rules that use the `authenticate()` or `authenticate.force( )` properties, the `deny.unauthorized( )` property is equivalent to `exception(authorization_failed)`. If policy does not contain any rules that require authentication, `deny.unauthorized( )` is equivalent to `exception(policy_denied)`.

The identity of the exception being returned can be tested in an `<Exception>` layer using `exception.id=`.

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Applies to HTTP transactions. For other protocols, the property is the equivalent to `deny( )`.

See Also

Conditions: `exception.id=`

Properties: `deny( )`, `exception( )`, `force_deny( )`, `force_exception( )`

direct( )

Used to prevent requests from being forwarded to a parent proxy or SOCKS server, when the ProxySG is configured to forward requests.

When set to `yes`, `<Forward>` layer policy is not evaluated for the transaction.

Syntax

```
direct(yes|no)
```

The default value is `no`, which allows request forwarding.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Cache>` layers.
- Does not apply to FTP over HTTP or transparent FTP transactions.

See Also

- Properties: `bypass_cache( )`, `dynamic_bypass`, `force_cache()`, `forward( )`, `reflect_ip( )`

dynamic_bypass( )

Used to indicate that a particular transparent request is not to be handled by the proxy, but instead be subjected to Proxy*SG* dynamic bypass methodology.

The `dynamic_bypass(yes)` property takes precedence over `authenticate();` however, a committed denial takes precedence over `dynamic_bypass(yes)`.

Syntax

```
dynamic_bypass(yes|no)
```

The default value is `no`.

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Applies to transparent HTTP transactions only.

See Also

- Properties: `advertisement( ), always_verify( ), bypass_cache( ), cache( ), cookie_sensitive( ), delete_on_abandonment( ), direct( ), force_cache(), pipeline( ), refresh( ), ttl( ), ua_sensitive( )`

exception( )

Selects a built-in or user-defined response to be returned to the user.

The `exception( )` property is overridden by `allow` or `deny( )`. To set an exception that cannot be overridden by `allow`, use `force_exception( )`.

The identity of the exception being returned can be tested in an `<Exception>` layer using `exception.id=`.

Note: When the exception response selected would have a Content-Length of 512 or fewer bytes, Internet Explorer may substitute "friendly" error messages. To prevent this behaviour use `exception.autopad(yes)`.

Syntax

```
exception(exception_id, details)
```

where:

- *`exception_id`*—Either the name of a built-in exception (refer to Chapter 14: "Advanced Policy" in the ProxySG *Configuration and Management Guide* for the list of built-in exceptions), or a name of the form `user_defined.`*`exception_id`* that refers to a user-defined exception page.
- *`details`*—A text string that is substituted for `$(exception.details)` within the selected exception.

Layer and Transaction Notes

- Use in `<Cache>`, `<Proxy>`, and `<Admin>` layers.
- Applies to all transactions.

See Also

- Conditions: `exception.id=`
- Properties: `allow`, `deny( )`, `deny.unauthorized( )`, `exception.autopad( )`, `force_deny( )`, `force_exception( )`

exception.autopad( )

Pad an HTTP exception response by including trailing whitespace in the response body so that Content-Length is at least 513 characters.

A setting of `yes` is used to prevent Internet Explorer from substituting *friendly* error messages in place of the exception response being returned, when the exception as configured would have a Content-Length of less than 512 characters.

Syntax

```
exception.autopad(yes|no)
```

where:

- `yes`—Enables auto-padding.
- `no`—Disables auto-padding.

The default value is `yes`.

Layer and Transaction Notes

- Use in `<Exception>` layers only.
- Applies to HTTP transactions.

See Also

- Conditions: `exception.id=`
- Properties: `exception( )`, `force_exception( )`

force_cache( )

Used to force caching of HTTP responses that would otherwise be considered uncacheable. The default HTTP caching behavior is restored using `force_cache(no)`. The value of the `force_cache( )` property is ignored unless all of the following property settings are in effect: `bypass_cache(no)`, `cache(yes)`, `cookie_sensitive(no)`, and `ua_sensitive(no)`.

Syntax

```
force_cache(all|no)
```

The default value is `no`.

Layer and Transaction Notes

- Use only in `<Cache>` layers.
- Applies to proxy transactions, which execute both `<Cache>` and `<Proxy>` layers.

Example

```
; Ensure objects at this URL are cached.
url=http://www.example.com/docs force_cache(all)
```

See Also

- Properties: `advertisement( )`, `always_verify( )`, `bypass_cache( )`, `cache( )`, `cookie_sensitive( )`, `dynamic_bypass`, `pipeline( )`, `refresh( )`, `ttl( )`, `ua_sensitive( )`

force_deny( )

The `force_deny( )` property is similar to `deny( )` except that it:

- Cannot be overridden by an allow.
- Overrides any pending termination (that is, if a `deny( )` has already been matched, and a `force_deny` or `force_exception` is subsequently matched, the latter commits.
- Commits immediately (that is, the first one matched applies).

The `force_deny( )` property is equivalent to `force_exception(policy_denied)`.

Syntax

```
force_deny
force_deny(details)
```

where *`details`* is a text string that will be substituted for `$(exception.details)` within the `policy_denied` exception. The *`details`* string may also contain CPL substitution patterns.

Layer and Transaction Notes

- Use in `<Cache>`, `<Proxy>`, and `<Admin>` layers.
- Do not use in `<Forward>` layers.
- Applies to all transactions.

See Also

- Conditions: `exception.id=`
- Properties: `deny()`, `force_exception()`

force_exception( )

The `force_exception( )` property is similar to exception except that it:

- Cannot be overridden by an `allow`.
- Overrides any pending termination (that is, if a `deny( )` has already been matched, and a `force_deny( )` or `force_exception( )` is subsequently matched, the latter commits.
- Commits immediately (that is, the first one matched applies).

Syntax

```
force_exception(exception_id)
force_exception(details)
```

where *`details`* is a text string that will be substituted for `$(exception.details)` within the specified exception. The *`details`* string may also contain CPL substitution patterns.

Layer and Transaction Notes

- Use in `<Cache>`, `<Proxy>`, and `<Admin>` layers.
- Applies to all transactions.

See Also

- Conditions: `exception.id=`
- Properties: `deny( )`, `exception( )`, `exception.autopad( )`, `force_deny( )`

force_patience_page( )

This property provides control over the application of the default patience page logic.

Syntax

```
force_patience_page(yes|no)
force_patience_page(reason )
force_patience_page.reason(yes|no)
force_patience_page[reason, ...](yes|no)
```

where:

`reason`—Takes one of the following values, corresponding to the overridable portions of the default logic that suppresses patience pages.

- `user-agent`—Overrides the suppression of patience pages for non-graphical browsers (any user agent string beginning with *mozilla* or *opera* is considered graphical).
- `extension`—Overrides the suppression of patience pages for graphical file extensions or extensions indicating cascading stylesheets, javscript, vbscript, vbx, or java applet, or flash animation content.
- `content-type`—Overrides the suppression of patience pages for content similar to that listed under `extension`, but based on the content-type header of the HTTP response.

The default is `force_patience_page(no)`.

Discussion

Each of the syntax variants has a different role in selecting the portions of patience page logic that will be overridden for the transaction:

- `force_patience_page(yes|no)` sets `(yes)` or clears `(no)` all reasons.
- `force_patience_page(reason, ..)` sets the listed reasons and clears any reasons not listed.
- `force_patience_page.reason( )` sets `(yes)` or clears `(no)` the specified reason.
- `force_patience_page.[reason, ...]( )` sets `(yes)` or clears `(no)` the listed reasons.

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Applies to HTTP proxy transactions.

See Also

- Properties: `patience_page( )`

forward( )

Determines forwarding behavior.

There is a box-wide configuration setting (`config>forwarding>sequence`) for the default forwarding failover sequence. The `forward( )` property is used to override the default forwarding failover sequence with a specific list of host and/or group aliases. The list of aliases might contain the special token `default`, which expands to include the default forward failover sequence defined in configuration.

Duplication is allowed in the specified alias list only in the case where a host or group named in the default failover sequence is also named explicitly in the alias_list.

In addition, there is a box-wide configuration setting (`config>forwarding>failure-mode`) for the default forward failure mode. The `forward.fail_open( )` property overrides the configured default.

Syntax

`forward(`*`alias_list`*`|no)`

where:

- *`alias_list`*—Forward this request through the specified alias list, which might refer to both forward hosts and groups. The ProxySG attempts to forward this request through the specified hosts or groups, in the order specified by the list. It proceeds to the next alias as necessary when the current host or group is down, as determined by health checks.
- `no`—Do not forward this request through a forwarding host. A SOCKS gateway or ICP host may still be used, depending on those properties. If neither are set, the request is sent directly to the origin server. Note that `no` overrides the default sequence defined in configuration.

The default value is `default`, as the only token in the *`alias_list`*.

Layer and Transaction Notes

- Use only in `<Forward>` layers.
- Applies to all transactions except administrator, instant messaging, and SOCKS.

See Also

- Properties: `direct( )`, `dynamic_bypass( )`, `icp( )`, `reflect_ip( )`, `refresh( )`, `socks_gateway( )`, `socks_gateway.fail_open( )`, `streaming.transport( )`

forward.fail_open( )

Controls whether the Proxy*SG* terminates or continues to process the request if the specified forwarding host or any designated backup or default cannot be contacted.

There is a box-wide configuration setting (`config>forwarding>failure-mode`) for the default forward failure mode. The `forward.fail_open( )` property overrides the configured default.

Syntax

```
forward.fail_open(yes|no)
```

where:

- `yes`—Continue to process the request if the specified forwarding host or any designated backup or default cannot be contacted. This may result in the request being sent through a SOCKS gateway or ICP, or may result in the request going directly to the origin server.
- `no`—Terminate the request if the specified forwarding host or any designated backup or default cannot be contacted.

The default value is `no`.

Layer and Transaction Notes

- Use only in `<Forward>` layers.
- Applies to all transactions except administrator, instant messaging, and SOCKS.

See Also

- Properties: `bypass_cache( )`, `dynamic_bypass`, `forward( )`, `reflect_ip( )`, `socks_gateway( )`, `socks_gateway.fail_open( )`

ftp.server_connection( )

Determines when the control connection to the server is established. If set to `deferred`, the proxy defers establishing the control connection to the server.

Syntax

```
ftp.server_connection(deferred|immediate)
```

The default value is `immediate.`

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Applies to FTP transactions.

See Also

- Properties: `ftp.server_data( )`, `ftp.transport( )`

ftp.server_data( )

Determines the type of data connection to be used with this FTP transaction.

Syntax

```
ftp.server_data(auto|passive|port)
```

where:

- `auto`—First attempt a PASV data connection. If this fails, switch to PORT.
- `passive`—Use a PASV data connection. PASV data connections are not allowed by some firewalls.
- `port`—Use a PORT data connection. FTP servers can be configured to not support PORT connections.

Layer and Transaction Notes

- Use in `<Forward>` layers.
- Applies to FTP transactions.

See Also

- Properties: `ftp.server_connection( )`, `ftp.transport( )`

ftp.transport( )

Determines the upstream transport mechanism.

This setting is not definitive. It depends on the capabilities of the selected forwarding host.

Syntax

```
ftp_transport(auto|ftp|http)
```

The default value is `auto`.

where:

- `auto`—Use the default transport for the upstream connection, as determined by the originating transport and the capabilities of any selected forwarding host.
- `ftp`—Use FTP as the upstream transport mechanism.
- `http`—Use HTTP as the upstream transport mechanism.

Layer and Transaction Notes

- Use in `<Forward>` layers.
- Applies only to WebFTP transactions where the client uses the HTTP protocol to request a URL with an `ftp:` schema.

See Also

- Properties: `ftp.server_connection( )`, `ftp.server_data( )`

http.force_ntlm_for_server_auth( )

Turns on/off NTLM cloaking on a per-request basis. Refer to Appendix A: "NTLM and CAASNT" in the Proxy*SG Configuration and Management Guide* for a discussion of NTLM cloaking.

Syntax

```
http.force_ntlm_for_server_auth(yes|no)
```

This property overrides the default specified in configuration.

where:

- `yes`—Enables NTLM cloaking.
- `no`—Disables NTLM cloaking.

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Applies to HTTP Proxy transactions.

http.request.version( )

The `http.request.version( )` property sets the version of the HTTP protocol to be used in the request to the origin content server or upstream proxy.

Syntax

```
http.request.version(1.0|1.1)
```

The default is taken from the CLI configuration setting `http version`, which can be set to either 1.0 or 1.1. Changing this value in the CLI changes the default for both `http.request.version( )` and `http.response.version( )`.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Cache>` layers.
- Applies to HTTP transactions.

See Also

- Conditions: `http.request.version=`
- Properties: `http.response.version( )`

http.response.version( )

The `http.response.version( )` property sets the version of the HTTP protocol to be used in the response to the client's user agent.

Syntax

```
http.response.version(1.0|1.1)
```

The default is taken from the CLI configuration setting `http version`, which can be set to either 1.0 or 1.1. Changing this value in the CLI changes the default for both `http.request.version( )` and `http.response.version( )`.

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Applies to HTTP transactions.

See Also

- Conditions: `http.response.version=`
- Properties: `http.request.version( )`

icp( )

Determines whether to consult ICP when forwarding requests. Any forwarding host or SOCKS gateway identified as an upstream target takes precedence over consulting ICP.

Syntax

```
icp(yes|no)
```

The default is `yes` if ICP hosts are configured, `no` otherwise.

where:

- `yes`—Consult ICP unless `forward( )` or `socks_gateway( )` properties are set. If no ICP hosts are configured, `yes` has no effect.
- `no`—Do not consult ICP hosts, even if configured.

Layer and Transaction Notes

- Use in `<Forward>` layers.
- Applies to all but SOCKS transactions.

See Also

- Properties: `direct( )`, `forward( )`, `reflect_ip( )`, `socks_gateway( )`

im.strip_attachments( )

Determines whether attachments are stripped from instant messages. If set to `yes`, attachments are stripped from instant messages.

Syntax

```
im.strip_attachments(yes|no)
```

The default value is `no`.

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Applies to instant messaging transactions.

See Also

- Conditions: `im.buddy_id=`, `im.chat_room.conference=`, `im.chat_room.id=`, `im.chat_room.invite_only=`, `im.chat_room.type=`, `im.chat_room.member=`, `im.chat_room.voice_enabled=`, `im.file.extension=`, `im.file.name=`, `im.file.path=`, `im.file.size=`, `im.message.route=`, `im.message.size=`, `im.message.text=`, `im.message.type=`, `im.method=`, `im.user_id=`

integrate_new_hosts( )

Determines whether to add new host addresses to health checks and load balancing.

Syntax

```
integrate_new_hosts(yes|no)
```

The default is `no`. If it is set to `yes`, any new host addresses encountered during DNS resolution of forwarding hosts are added to health checks and load balancing.

Layer and Transaction Notes

- Use in `<Forward>` layers.
- Applies to everything but SOCKS and administrator transactions.

See Also

- Properties: `forward( )`

label( )

This deprecated property is provided for backward compatibility with CacheOS 4.x filter files. For more information, see "action( )" on page 156.

log.rewrite.*field-id*( )

The `log.rewrite.field-id` property controls rewrites of a specific log field in one or more access logs. Individual access logs are referenced by the name given in configuration. Configuration also determines the format of the each log. For more information on logging, refer to Chapter 19: "Access Logging" in the ProxySG *Configuration and Management Guide*.

Syntax

```
log.rewrite.field-id("substitution"|no)
log.rewrite.field-id[log_name_list]("substitution"|no)
```

where:

- *field-id*—Specifies the log field to rewrite. Some *field-ids* have embedded parentheses, for example `cs(User-agent)`. These field-ids must be enclosed in quotes. There are two choices for quoting, either of which are accepted by the CPL compiler:

  ```
  log.rewrite."cs(User-agent)"(...)
  "log.rewrite.cs(User-agent)(...)"
  ```

  Either single or double quotes may be used.

- *log_name_list*—A comma separated list of configured access logs, of the form:

  ```
  log_name_1, log_name_2, ...
  ```

- *substitution*—A quoted string containing replacement text for the field. The substitution string can contain CPL substitution variables.

- `no`—Cancels any previous substitution for this log field.

Discussion

Each of the syntax variants has a different role in specifying the rewrites for the access log fields used to record the transaction:

- `log.rewrite.field-id( )` specifies a rewrite of the *field_id* field in all access logs selected for this transaction.

- `log.rewrite.field-id[log_name_list]( )` specifies a rewrite of the *field_id* field in all access logs named in `log_name_list`. The *field_id* field in any logs not named in the list is unaffected.

Layer and Transaction Notes

- Use in all layers.

- Applies to all proxy transactions.

See Also

- Properties: `access_log( )`, `log.suppress.field-id()`

log.suppress.*field-id*( )

The `log.suppress.`*`field-id`*`( )` property controls suppression of the specified *`field-id`* in one or more access logs. Individual access logs are referenced by the name given in configuration. Configuration also determines the format of the each log. For more information on logging, refer to Chapter 19: "Access Logging" in the Proxy*SG Configuration and Management Guide*.

Syntax

```
log.suppress.field-id(yes|no)
log.suppress.field-id[log_name_list](yes|no)
```

where:

- *`field-id`*—Specifies the log field to suppress. Some *`field-ids`* have embedded parentheses, for example `cs(User-agent)`. These field-ids must be enclosed in quotes. There are two choices for quoting, either of which are accepted by the CPL compiler:

  ```
  log.suppress."cs(User-agent)"(yes|no)
  "log.suppress.cs(User-agent)(yes|no)"
  ```

  Either single or double quotes may be used.

- *`log_name_list`*—A comma separated list of configured access logs, of the form:

  `log_name_1, log_name_2, ...`

- `yes`— Suppresses the specified *`field-id`*
- `no`—Turns suppression off for the specified *`field-id`*

Discussion

Each of the syntax variants has a different role in suppressing the access log fields used to record the transaction:

- `log.suppress.`*`field-id`*`( )` controls suppression of the *`field_id`* field in all access logs selected for this transaction.
- `log.suppress.`*`field-id`*`[`*`log_name_list`*`]( )` controls suppression of the *`field_id`* field in all access logs named in `log_name_list`. The *`field_id`* field in any logs not named in the list is unaffected.

Layer and Transaction Notes

- Use in all layers.
- Applies to all proxy transactions.

See Also

- Properties: `access_log( )`, `log.rewrite.`*`field-id`*`( )`

max_bitrate( )

Enforces upper limits on the instantaneous bandwidth of the current streaming transaction. This policy is enforced during initial connection setup. If the client requests a higher bit rate than allowed by policy, the request is denied.

Note: Under certain network conditions, a client may receive a stream that temporarily exceeds the specified bit rate.

Replaces: `max_bitrate(no) replaces max_bitrate(0)`

Syntax

```
max_bitrate(bitrate|no)
```

The default value is `no`.

where:

- *`bitrate`*—Maximum bit rate allowed. Specify using an integer, in bits, kilobits (1000x), or megabits (1,000,000x), as follows: *`integer`* `|` *`integer`*`k` `|` *`integer`*`m`.
- `no`—Allows any bitrate.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Cache>` layers.
- Applies to streaming transactions.

Example

```
; Client bit rate for streaming media cannot exceed 56 kilobits.
max_bitrate(56k)
```

See Also

- Conditions: `bitrate=`, `live=`, `streaming.content=`

never_refresh_before_expiry( )

The `never_refresh_before_expiry( )` property is similar to the CLI command:

```
SGOS#(config) http strict-expiration refresh
```

except that it provides per-transaction control to allow overriding the box-wide default set by the command.

Syntax

```
never_refresh_before_expiry(yes|no)
```

The default value is taken from configuration.

Layer and Transaction Notes

- Use in `<Cache>` layers.
- Applies to proxy transactions.

See Also

- Properties: `never_serve_after_expiry( )`, `remove_IMS_from_GET( )`, `remove_PNC_from_GET( )`, `remove_reload_from_IE_GET( )`
- The Proxy*SG Command Line Reference* for information on the `http strict-expiration` command.

never_serve_after_expiry( )

The `never_serve_after_expiry( )` property is similar to the CLI command:

```
SGOS#(config) http strict-expiration serve
```

except that it provides per transaction control to allow overriding the box-wide default set by the command.

Syntax

```
never_serve_after_expiry(yes|no)
```

The default value is taken from configuration.

Layer and Transaction Notes

- Use in `<Cache>` layers.
- Applies to proxy transactions.

See Also

- Properties: `always_verify()`, `never_refresh_before_expiry()`
- The ProxySG *Command Line Reference* for information on the `http strict-expiration` command.

patience_page( )

Controls whether or not a patience page can be served, and if so, the delay interval before serving.

If no `patience_page` property is explicitly set, the decision about whether to serve a patience page and the delay before a patience page is presented are taken from the ICAP service configuration (but are still subject to default patience page policy). To control the application of default patience page policy, use `force_patience_page( )`.

Syntax

```
patience_page(no|delay)
```

The default value is taken from configuration.

where:

- `no`—A patience page will not be served.
- `delay` —(number of seconds, in the range 5-65535). Subject to default patience page policy, a patience page is served after the specified number of seconds.

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Applies to HTTP proxy transactions only.

See Also

- Properties: `force_patience_page( )`

pipeline( )

Determines whether an object embedded within an HTML container object is pipelined. Set to `yes` to force pipelining, or set to `no` to prevent the embedded object from being pipelined. Note that this property affects processing of the individual URLs embedded within a container object. It does not prevent parsing of the container object itself.

If this property is used with a URL access condition, such as `url.host=`, each embedded object on a page is evaluated against that policy rule to determine pipelining behavior. For example, a rule that disallows pipelining for a particular host would still allow pipelining for images on the host's pages that come from other hosts.

Replaces: `prefetch()`

Syntax

```
pipeline(yes|no)
```

The default value is `yes`.

Layer and Transaction Notes

- Use in `<Cache>` layers.
- Applies to HTTP proxy transactions.

prefetch( )

This deprecated property has been replaced by `pipeline( )`. For more information, see "pipeline( )" on page 202.

reflect_ip( )

Determines how the client IP address is presented to the origin server for explicitly proxied requests.

Replaces:

- `reflect_ip(vip)` replaces `reflect_vip(yes)`.
- `reflect_ip(auto)` replaces `reflect_vip(no)`.

Syntax

```
reflect_ip(auto|no|client|vip|ip_address)
```

The default value is `auto`.

where:

- `auto`—Might reflect the client IP address, based on a config setting for spoofing.
- `no`—The appliance's IP address is used to originate upstream connections.
- `client`—The client's IP address is used in initiating upstream connections.
- `vip`—The appliance's VIP on which the client request arrived is used to originate upstream traffic.
- *`ip_address`*—A specific IP address, which must be an address (either physical or virtual) belonging to the ProxySG . If not, at runtime this is converted to `auto`.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Forward>` layers.
- Applies to proxy transactions.

Example

```
; For requests from a specific client, use the virtual IP address.
<proxy>
client.address=10.1.198.0 reflect_vip(yes)
```

See Also

- Properties: `forward( )`

reflect_vip( )

This deprecated syntax has been replaced by the `reflect_ip( )` property. For more information, see "reflect_ip( )" on page 204.

refresh( )

Controls refreshing of requested objects. Set to `no` to prevent refreshing of the object if it is cached. Set to `yes` to allow the cache to behave normally.

Syntax

```
refresh(yes|no)
```

The default value is `yes`.

Layer and Transaction Notes

- Use in `<Cache>` layers.
- Do not use in `<Proxy>` layers.

See Also

- Properties: `advertisement( )`, `always_verify( )`, `bypass_cache( )`, `cache( )`, `cookie_sensitive( )`, `direct( )`, `force_cache( )`, `never_refresh_before_expiry( )`, `Never_serve_after_expiry( )`, `ttl( )`, `ua_sensitive( )`

remove_IMS_from_GET( )

The `remove_IMS_from_GET( )` property is similar to the CLI command:

```
SGOS#(config) http substitute if-modified-since
```

except that it provides per transaction control to allow overriding the box-wide default set by the command.

Syntax

```
remove_IMS_from_GET(yes|no)
```

The default value is taken from configuration.

Layer and Transaction Notes

- Use in `<Cache>` layers.
- Applies to HTTP proxy transactions.

See Also

- Properties: `never_refresh_before_expiry( )`, `never_serve_after_expiry( )`, `remove_PNC_from_GET( )`, `remove_reload_from_IE_GET( )`
- The Proxy*SG Command Line Reference* for information on the `http substitute` command.

remove_PNC_from_GET( )

The `remove_PNC_from_GET` property is similar to the CLI command:

```
SGOS#(config) http substitute pragma-no-cache
```

except that it provides per transaction control to allow overriding the box-wide default set by the command.

Syntax

```
remove_PNC_from_GET(yes|no)
```

The default value is taken from configuration.

Layer and Transaction Notes

- Use in `<Cache>` layers.
- Applies to HTTP proxy transactions.

See Also

- Properties: `never_refresh_before_expiry()`, `never_serve_after_expiry( )`, `remove_IMS_from_GET( )`, `remove_reload_from_IE_GET( )`
- The Proxy*SG Command Line Reference* for information on the `http substitute` command.

remove_reload_from_IE_GET( )

The `remove_reload_from_IE_GET( )` property is similar to the CLI command:

```
SGOS#(config) http substitute ie-reload
```

except that it provides per transaction control to override the box-wide default set by the command.

Syntax

```
remove_reload_from_IE_GET(yes|no)
```

The default value is taken from configuration.

Layer and Transaction Notes

- Use in `<Cache>` layers.
- Applies to HTTP proxy transactions.

See Also

- Properties: `never_refresh_before_expiry( )`, `never_serve_after_expiry( )`, `remove_IMS_from_GET( )`, `remove_PNC_from_GET( )`
- The Proxy*SG Command Line Reference* for information on the `http substitute` command.

request.filter_service( )

Controls whether the request is processed by an external content filter service. The ProxySG currently supports Websense Enterprise Server external content filtering.

Directing the request to an external content filter service does not affect policy based on categories determined through an on-box vendor or CPL category definitions.

Categories determined by Websense Enterprise Server are not available to the `category=` condition, although they appear in access logs. Effectively, all policy based on the Websense determined categories must be implemented on the Websense server.

Note: This property might be overridden by the deprecated `content_filter_override(yes)` property.

Replaces: `content_filter_override(yes)`

Syntax

```
request.filter_service(servicename[, fail_open|fail_closed])
request.filter_service(no)
```

The default values are `no` and `fail_closed`.

where:

- *`servicename`*—A configured external content filter service that supports request modification. Currently only Websense Enterprise Server is supported. On upgrade, the service name `websense` is automatically generated.
- `fail_open`—If *`servicename`* is unavailable, the request is processed and a response may be delivered, subject to other policy.
- `fail_closed`—If the *`servicename`* is unavailable, the request is denied.
- `no`—Prevents the request from being sent from the ProxySG to the external content filter service.

Layer and Transaction Notes

- Use in `<Cache>` and `<Proxy>` layers.
- Applies to FTP and HTTP transactions.

Example

The following example directs requests to the Websense server, but allows processing to continue if the service in unavailable:

```
<proxy>
   request_filter_service(websense, fail_open)
```

The following policy establishes a general rule that all request are processed by the external filter service named `filter`. It then specifies some exceptions to this general rule in a later layer:

```
<proxy> ; All request are content-filtered by default
   request.filter_service( filter )
<proxy> request.filter_servce( no )  ; exceptions to content-filtering
```

```
url.address=10.0.0.0/8  ; don't filter internal network
client.address=10.1.2.3  ; don't filter this client
```

See Also

- The Proxy*SG Command Line Reference* for information on configuring Websense off-box services.

request.icap_service( )

Determines whether a request from a client should be processed by an external ICAP service before going out. Typical applications include content filtering and virus scanning.

Syntax

```
request.icap_service(servicename [, fail_open | fail_closed])
request.icap_service(no)
```

The default values are `no` and `fail_closed`.

where:

- *`servicename`*—A configured ICAP service that supports request modification.
- `fail_open`—If the Proxy*SG* cannot communicate with the ICAP service, the request is processed and a response delivered (subject to other policies).
- `fail_closed` —If the Proxy*SG* cannot communicate with the ICAP service, the request is denied. This is the default and need not be specified to be in effect.
- `no`—Disables ICAP processing for this request, regardless of whether there is an ICAP service name defined in configuration. This is useful when ICAP processing is generally desired, but specific exceptions are required.

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Applies to FTP and HTTP transactions.

See Also

- Properties: `response.icap_service( )`

response.icap_service( )

Determines whether a response to a client request is first sent to an ICAP service before being given to the client. Depending on the ICAP service, the response may be allowed, denied, or altered. Typical applications include virus scanning.

Syntax

```
response.icap_service(servicename [, fail_open | fail_closed])
response.icap_service(no)
```

The default values are `no` and `fail_closed`.

where:

- *`servicename`*—A configured ICAP service that supports response modification.
- `fail_open` —If the Proxy*SG* cannot communicate with the ICAP service, the response may be delivered (subject to other policies).
- `fail_closed` —If the Proxy*SG* cannot communicate with the ICAP service, the request is denied. This is the default and need not be specified to be in effect.
- `response.icap_service (no)`—Disables ICAP processing for this response, regardless of whether there is an ICAP service name defined in configuration. This is useful when ICAP processing is generally desired, but specific exceptions are required.

Layer and Transaction Notes

- Use in `<Cache>` layers.
- Applies to HTTP, FTP, proxy, and cache transactions.

See Also

- Properties: `request.icap_service( )`

service( )

This deprecated syntax has been replaced by the `allow`, `deny( )` and `exception( )` properties.

socks.accelerate( )

The `socks.accelerate` property controls the SOCKS proxy handoff to other protocol agents.

Syntax

```
socks.accelerate(no|auto|http|aol_im|msn_im|yahoo_im)
```

The default value is `auto`.

where:

- `no`—The SOCKS proxy does not hand off the transaction to another proxy agent, but tunnels the SOCKS transaction.
- `auto`—The handoff is determined by the URL scheme.

Any other value forces the SOCKS proxy to hand off the transaction to the agent for the indicated protocol.

The `socks.accelerated=` condition can be used to test which agent was selected for handoff. The `tunneled=` condition can be used to test for unaccelerated (tunneled) SOCKS transactions.

After the handoff, the transaction is subject to policy as a proxy transaction for the appropriate protocol. Within that policy, the `socks=` condition can be used to test for transactions use SOCKS for client communication.

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Applies to SOCKS proxy transactions.

See Also

- Properties: `socks_gateway( )`, `socks.authenticate( )`, `socks.authenticate.force( )`
- Conditions: `socks=`, `socks.accelerated=`, `socks.destination_address=`, `socks.destination_port=`, `socks.method=`, `socks.tunneled=`, `socks.version=`

socks.authenticate( )

The same realms can be used for SOCKS proxy authentication as can be used for regular proxy authentication. This form of authentication applies only to SOCKS transactions.

The regular `authenticate( )` property does not apply to SOCKS transactions. However, if an accelerated SOCKS transaction has already been authenticated in the same realm by the SOCKS proxy, no new authentication challenge is issued. If the realms identified in the `socks.authenticate( )` and `authenticate( )` properties differ, however, a new challenge is issued by the proxy agent used to accelerate the SOCKS transaction.

Note: There is no optional display name.

Following SOCKS proxy authentication, the standard `user=`, `group=`, and `realm=` tests are available.

The relation between SOCKS authentication and denial is controlled through the `socks.authenticate.force( )` property. The default setting `no` implies that denial overrides `socks.authenticate( )`, with the result that user names may not appear for denied requests if that denial could be determined without authentication. To ensure that user names appear in access logs, use `socks.authenticate.force(yes)`.

Syntax

```
socks.authenticate(realmname)
```

where:

- *realmname*—One of the already-configured realms.
- Consider that `socks.authenticate()` depends exclusively on a limited number of triggers:
  - `proxy_address=`
  - `proxy_card=`
  - `proxy_port=`
  - `client_address=`
  - `socks.version=`

  Date and time triggers, while available, are not recommended.

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Applies to SOCKS proxy transactions.

See Also

- Properties: `authenticate()`, `socks_gateway()`, `socks.accelerate()`, `socks.authenticate.force()`
- Conditions: `socks=`, `socks.destination_address=`, `socks.destination_port=`, `socks.method=`, `socks.tunneled=`, `socks.version=`

socks.authenticate.force( )

This property controls the relation between SOCKS authentication and denial.

Syntax

```
socks.authenticate.force(yes|no)
```

The default value is `no`.

where:

- `yes`—Makes `socks.authenticate( )` higher priority than `deny( )` or `exception( )`. Use `yes` to ensure that userID's are available for access logging, even of denied requests.
- `no`—`deny( )` and `exception( )` have a higher priority than `socks.authenticate( )`. This setting allows early denial (based on proxy card, address or port, client address, or SOCKS version, for example). That is, the denial preempts any authentication requirement.

Note: This does not affect regular `authenticate( )`.

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Applies to SOCKS proxy transactions.

See Also

- Properties: `socks.authenticate( )`, `socks_gateway( )`, `socks.accelerate( )`
- Conditions: `socks.destination_address=`, `socks.destination_port=`, `socks.method=`, `socks.tunneled=`, `socks.version=`

socks_gateway( )

Controls whether or not the request associated with the current transaction is sent through a SOCKS gateway.

There is a box-wide configuration setting (`config>socks-gateways>sequence`) for the default SOCKS gateway failover sequence. The `socks_gateway( )` property is used to override the default SOCKS gateway failover sequence with a specific list of SOCKS gateway aliases. The list of aliases might contain the special token `default`, which expands to include the default SOCKS gateway failover sequence defined in configuration.

Duplication is allowed in the specified alias list only in the case where a gateway named in the default failover sequence is also named explicitly in `alias_list`.

In addition, there is a box-wide configuration setting (`config>socks-gateways>failure-mode`) for the default SOCKS gateway failure mode. The `socks_gateway.fail_open( )` property overrides the configured default.

Syntax

```
socks_gateway(alias_list|no)
```

The default value is `no`.

where:

- *`alias_list`*—Send this request through the specified alias list. The Proxy*SG* attempts to send this request through the specified gateways in the order specified by the list. It proceeds to the next gateway alias as necessary when the gateway is down, as determined by health checks.
- `no`—Do not send this request through a SOCKS gateway. A forwarding host or ICP host may still be used, depending on those properties. If neither are set, the request is sent directly to the origin server. A setting of `no` overrides the default sequence defined in configuration.

Layer and Transaction Notes

- Use in `<Forward>` layers.
- Applies to all except administrator transactions.

See Also

- Properties: `direct( )`, `forward( )`, `socks.accelerate( )`, `socks.authenticate( )`, `socks.authenticate.force( )`
- Conditions: `socks.destination_address=`, `socks.destination_port=`, `socks.method=`, `socks.tunneled=`, `socks.version=`

socks_gateway.fail_open( )

Controls whether the Proxy*SG* terminates or continues to process the request if the specified SOCKS gateway or any designated backup or default cannot be contacted.

There is a box-wide configuration setting (`config>socks-gateways>failure-mode`) for the default SOCKS gateway failure mode. The `socks_gateway.fail_open( )` property overrides the configured default.

Syntax

```
socks_gateway.fail_open(yes|no)
```

The default value is `no`.

where:

- `yes`—Continue to process the request if the specified SOCKS gateway or any designated backup or default cannot be contacted. This may result in the request being forwarded through a forwarding host or ICP, or may result in the request going direct to the origin server.
- `no`—Terminates the request if the specified SOCKS gateway or any designated backup or default cannot be contacted.

Layer and Transaction Notes

- Use in `<Forward>` layers.
- Applies to all except administrator transactions.

See Also

- Properties: `socks.accelerate( )`, `socks.authenticate( )`, `socks.authenticate.force( )`, `socks_gateway( )`
- Conditions: `socks.destination_address=`, `socks.destination_port=`, `socks.method=`, `socks.tunneled=`, `socks.version=`

streaming.transport( )

Determines the upstream transport mechanism to be used for this streaming transaction. This setting is not definitive. The ability to use the specified transport mechanism depends on the capabilities of the selected forwarding host.

Syntax

```
streaming.transport(auto|tcp|http)
```

where:

- `auto`—Use the default transport for the upstream connection, as determined by the originating transport and the capabilities of any selected forwarding host.
- `tcp`—Use TCP as the upstream transport mechanism.
- `http`—Use HTTP as the upstream transport mechanism.

Layer and Transaction Notes

- Use in `<Forward>` layers.
- Applies to streaming transactions.

See Also

- Conditions: `bitrate=`, `live=`, `streaming.client=`, `streaming.content=`

terminate_connection( )

The `terminate_connection( )` property is used in an `<Exception>` layer to drop the connection rather than return the exception response. The `yes` option terminates the connection instead of returning the response. (This property provides backwards compatible support with the TERMINATE_CONNECTION error pages directive supported in SGOS 2.x.)

Syntax

```
terminate_connection(yes|no)
```

The default is `no`.

Layer and Transaction Notes

- Use in `<Exception>` layers.
- Applies to HTTP transactions.

trace.destination( )

Used to change the default path to the trace output file. By default, policy evaluation trace output is written to an object in the cache accessible using a console URL of the following form:

`http://ProxySG_IP_address:8081/Policy/Trace/path`

Syntax

`trace.destination(path)`

where *path* is, by default, `default_trace.html`. You can change *path* to a filename or directory path, or both. If only a directory is provided, the default trace filename is used.

Layer and Transaction Notes

- Use in any layer.
- Applies to all transactions.

Example

```
; Change directory location of trace output file to
; http://ProxySG_IP_address:8081/Policy/Trace/test/default_trace.html
trace.destination(test/)


; Change trace output file location to
; http://ProxySG_IP_address:8081/Policy/Trace/test/phase_2.html
trace.destination(test/phase_2.html)
```

See Also

- Properties: `trace.request()`, `trace.rules()`

trace.request( )

Determines whether detailed trace output is generated for the current request. The default value is no, which produces no output. Trace output is generated at the end of a request, and includes request parameters, property settings, and the effects of all actions taken. Output tracing can be set conditionally by creating a rule that combines this property with conditions such as `url=` or `client.address=`.

By default, trace output is written to an object accessible using the following console URL:

`http://ProxySG_IP_address:8081/Policy/Trace/default_trace.html`

The trace output location can be controlled using the `trace.destination( )` property.

Note: Tracing is best used temporarily, such as for troubleshooting; the `log_message( )` action is best for on-going monitoring.

Syntax

```
trace.request(yes|no)
```

The default value is no.

Layer and Transaction Notes

- Use in any layer.
- Applies to all transactions.

Example

```
; Generate trace details when a specific URL is requested.
url=//www.example.com/confidential trace.request(yes)
```

See Also

- Properties: `trace.destination()`, `trace.rules()`

trace.rules( )

Determines whether trace output is generated showing policy rule evaluation for the transaction.

By default, trace output is written to an object accessible using the following console URL:

```
http://ProxySG_IP_address:8081/Policy/Trace/default_trace.html
```

The trace output location can be controlled using the `trace.destination( )` property.

Note: Tracing is best used temporarily, such as for troubleshooting; the `log_message( )` action is best for on-going monitoring.

Syntax

```
trace.rules(yes|no|all)
```

where:

- `yes`—Generates output only for rules that match the request.
- `all`–Additionally shows which rules were skipped because one or more of their conditions were false or not applicable to the current transaction; displays the specific condition in the rule that failed.
- `no`–Suppresses output associated with policy rule evaluation.

The default value is `no`.

Layer and Transaction Note

- Use in `<Cache>` and `<Forward>` layers.

Example

```
; Generate trace messages.
<proxy>
trace.rules(yes) trace.request(yes)
```

See Also

- Properties: `trace.destination()`, `trace.request()`

ttl( )

Sets the time-to-live (TTL) value of an object in the cache, in seconds. Upon expiration, the cached copy is considered stale and will be re-obtained from the origin server when next accessed. However, this property has an effect only if the following HTTP command line option is enabled: `Force explicit expirations: Never serve after.`

If the above option is not set, the Proxy*SG*'s freshness algorithm determines the time-to-live value.

Note: `advertisement(yes)` overrides any `ttl( )` value.

Syntax

`ttl(`*`seconds`*`)`

where *`seconds`* is an integer, specifying the number of seconds an object remains in the cache before it is deleted. The maximum value is 4294967295, or about 136 years.

The default value is specified by configuration.

Layer and Transaction Notes

- Use in `<Cache>` layers.
- Do not use in `<Proxy>` layers.

Example

```
; Delete the specified cached objects after 30 seconds.
url=//www.example.com/dyn_images ttl(30)
```

See Also

- Properties: `advertisement( ), cache( )`

ua_sensitive( )

Used to modify caching behavior by declaring that the response for a given object is expected to vary based on the user agent used to retrieve the object. Set to `yes` to specify this behavior.

Using `ua_sensitive(yes)` has the same effect as `cache(no)`.

Note: Remember that any conflict among CPL property settings is resolved by CPL's evaluation logic, which uses the property value that was last set when evaluation ends.

Syntax

```
ua_sensitive(yes|no)
```

The default value is `no`.

Layer and Transaction Notes

- Use in `<Cache>` layers.
- Do not use in `<Proxy>` layers.
- Applies to proxy transactions, which execute both `<Cache>` and `<Proxy>` layers.
- Does not apply to FTP over HTTP transactions.

See Also

Properties: `advertisement( )`, `always_verify( )`, `bypass_cache( )`, `cache( )`, `cookie_sensitive( )`, `delete_on_abandonment( )`, `direct( )`, `dynamic_bypass( )`, `force_cache( )`, `pipeline( )`, `refresh( )`, `ttl( )`

Chapter 5: *Action Reference*

An *action* takes arguments and is wrapped in a user-named action definition block. When the action definition is called from a policy rule, any actions it contains operate on their respective arguments. Within a rule, named action definitions are enabled and disabled using the `action( )`property.

Actions take the following general form:

```
action(argument1, ...)
```

An action block is limited to the common subset among the allowed layers of each of the actions it contains. Actions appear only within action definitions. They cannot appear in `<Admin>` layers.

Argument Syntax

The allowed syntax for action arguments depends on the action.

- String—A string argument must be quoted if it contains whitespace or other special characters. For example: `log_message("Access alert")`.
- Enumeration—Actions such as `delete( )` use as an argument a token specifying the transaction component on which to act. For example: a header name such as `request_header.Referer`.
- Regular expression—Several actions take regular expressions. For more information about writing regular expressions, refer to Appendix E, "Using Regular Expressions," in the Blue Coat Proxy*SG Configuration and Management Guide*.
- Variable substitution—The quoted strings in some action arguments can include variable substitution substrings. These include the various versions of the replacement argument of the `redirect( )`, `rewrite( )`, and `rewrite( )` actions, and the string argument in the `append( )`, `log_message( )`, and `set(header, string)` actions. A variable substitution is a substring of the form:

  `$(name)`

  where *name* is one of the allowed substitution variables.

  For a complete list of substitutions, see Appendix D: "CPL Substitutions".

Action Reference

The remainder of this chapter lists the actions and their accepted values. It also provides the context in which each action can be used and examples of how to use them.

append( )

Appends a new component to the specified header.

Note: An error results if two header modification actions modify the same header. This results in a compile time error if the conflicting actions are within the same action definition block. A runtime error is recorded in the event log if the conflicting actions are defined in different blocks.

Syntax

```
append(header, string)
append(im.message.text, string)
```

where:

- *header*—A header specified using the following form. For a list of recognized headers, including headers that support field repetition, see Appendix C.
  - `request.header.`*header_name*—Identifies a recognized HTTP request header.
  - `response.header.`*header_name*—Identifies a recognized HTTP response header.
  - `request.x_header.`*header_name*—Identifies any request header, including custom headers.
  - `response.x_header.`*header_name*—Identifies any response header, including custom headers.
- *string*—A quoted string that can optionally include one or more variable substitutions.
- `im.message.text`—Appends the specified *string* to the end of the instant message text.

Layer and Transaction Notes

- Do not use from `<Admin>` or `<Forward>` layers.
- Use from `<Proxy>` or `<Cache>` layers

See Also

- Actions: `delete( )`, `delete_matching( )`, `rewrite(`*header, regex_pattern, replacement_component*`)`, `set(`*header, string*`)`
- Conditions: `request.header.`*header_name*`=`, `request.header.`*header_name*`.address=`, `request.x_header.`*header_name*`=`, `request.x_header.`*header_name*`.address=`, `response.header.`*header_name*`=`, `response.x_header.`*header_name*`=`

delete( )

Deletes all components of the specified header.

Note: An error results if two header modification actions modify the same header. The error is noted at compile time if the conflicting actions are within the same action definition block. A runtime error is recorded in the event log if the conflicting actions are defined in different blocks.

Syntax

```
delete(header)
```

where:

header—A header specified using the following form. For a list of recognized headers, see Appendix C.

- `request.header.`*`header_name`*—Identifies a recognized HTTP request header.
- `response.header.`*`header_name`*—Identifies a recognized HTTP response header.
- `request.x_header.`*`header_name`*—Identifies any request header, including custom headers.
- `response.x_header.`*`header_name`*—Identifies any response header, including custom headers.
- `exception.response.header.`*`header_name`*—Identifies a recognized HTTP response header from the exception response.

Layer and Transaction Notes

- Use with `exception.response.header.`*`header_name`* in `<Proxy>` or `<Exception>` layers.
- Use with request or response headers in `<Proxy>` or `<Cache>` layers.
- Do not use in `<Admin>` or `<Forward>` layers.
- Applies to HTTP transactions.

Example

```
; Delete the Referer request header, and also log the action taken.

define action DeleteReferer
   log_message("Referer header deleted: $(request.header.Referer)")
   delete(request.header.Referer)
end action DeleteReferer
```

See Also

- Actions: `append( )`, `delete_matching( )`, `rewrite(`*`header, regex_pattern, replacement_component`*`)`, `set(`*`header, string`*`)`
- Conditions: `request.header.`*`header_name`*`=`, `request.header.`*`header_name`*`.address=`, `request.x_header.`*`header_name`*`=`, `request.x_header.`*`header_name`*`.address=`, `response.header.`*`header_name`*`=`, `response.x_header.`*`header_name`*`=`

delete_matching( )

Deletes all components of the specified header that contain a substring matching a regular-expression pattern.

Note: An error results if two header modification actions modify the same header. The error is noted at compile time if the conflicting actions are within the same action definition block. A runtime error is recorded in the event log if the conflicting actions are defined in different blocks.

Syntax

```
delete_matching(header, regex_pattern)
```

where:

- *header*—A header specified using the following form. For a list of recognized headers, see Appendix C.
  - `request.header.`*header_name*— Identifies a recognized HTTP request header.
  - `response.header.`*header_name*—Identifies a recognized HTTP response header.
  - `request.x_header.`*header_name*—Identifies any request header, including custom headers.
  - `response.x_header.`*header_name*—Identifies any response header, including custom headers.
- *regex_pattern*—A quoted regular-expression pattern. For more information, refer to Appendix E, "Using Regular Expressions," in the *Blue Coat ProxySG Configuration and Management Guide*.

Layer and Transaction Notes

Do not use in `<Exception>`, `<Forward>`, or `<Admin>` layers.

See Also

- Actions: `append( )`, `delete( )`, `rewrite(`*header, regex_pattern, replacement_component*`)`, `set(`*header, string*`)`
- Conditions: `request.header.`*header_name=*, `request.header.`*header_name*`.address=`, `request.x_header.`*header_name=*, `request.x_header.`*header_name*`.address=`, `response.header.`*header_name=*, `response.x_header.`*header_name=*

im.alert( )

Deliver a message in-band to the instant messaging user. The *text* appears in the instant message window.

This action is similar to `log_message( )`, except that it appends entries to a list in the instant messaging transaction that the IM protocol renders in an appropriate way. Multiple alerts can be appended to a transaction. The protocol determines how multiple alerts appear to the user.

Syntax

```
im.alert(text)
```

where *text* is a quoted string that can optionally include one or more variable substitutions.

Layer and Transaction Notes

Use in `<Proxy>` and `<Cache>` layers.

See Also

- Actions: `log_message( )`

log_message( )

Writes the specified string to the ProxySG event log.

Events generated by `log_message( )` are viewed by selecting the Policy messages event logging level in the Management Console.

Note: This is independent of access logging.

Syntax

```
log_message(string)
```

Where *`string`* is a quoted string that can optionally include one or more variable substitutions.

Layer and Transaction Notes

Can be referenced by any layer.

Example

```
; Log the action taken, and include the original value of the Referer header.

define action DeleteReferer
   log_message("Referer header deleted: $(request.header.Referer)")
   delete(request.header.Referer)
end action DeleteReferer
```

See Also

- Actions: `notify_email( )`, `notify_snmp( )`
- Properties: `access_log( )`, `log.rewrite( )`, `log.suppress( )`

notify_email( )

Sends an email notification to the list of recipients specified in the Event Log mail configuration. The sender of the email appears as *Primary_ProxySG_IP_address - configured_appliance_hostname*>. You can specify multiple `notify_email` actions, which may result in multiple mail messages for a single transaction.

The email is sent when the transaction terminates. The email is sent to the list of recipients specified in the Event Log mail configuration.

Syntax

```
notify_email(subject, body)
```

where *subject* and *body* are quoted strings that can optionally include one or more variable substitutions.

Layer and Transaction Notes

Can be referenced by any layer.

Example

```
define condition restricted_sites
   url.domain=a_very_bad_site
   ...

end

<proxy>
   condition=restricted_sites action.notify_restricted(yes)

define action notify_restricted
   notify_email("restricted: ", \
               "$(client.address) accessed url: $(url)")
end
```

See Also

- Actions: `log_message( )`, `notify_snmp( )`

notify_snmp( )

Multiple `notify_snmp` actions may be specified, resulting in multiple SNMP traps for a single transaction.

The SNMP trap is sent when the transaction terminates.

Syntax

```
notify_snmp(message)
```

where *message* is a quoted string that can optionally include one or more variable substitutions.

Layer and Transaction Notes

Can be referenced by any layer.

See Also

- Actions: `log_message( ), notify_email( )`

redirect( )

Ends the current HTTP transaction and returns an HTTP redirect response to the client by setting the `policy_redirect` exception. Use this action to specify an HTTP 3*xx* response code, optionally set substitution variables based on the request URL, and generate the new Location response-header URL after performing variable substitution.

FTP over HTTP requests are redirected for Netscape Navigator clients, but not Microsoft Internet Explorer clients. To avoid this issue, do not use the `redirect( )` action when the `url.scheme=ftp` condition is true. For example, if the `http_redirect` action definition contains a `redirect( )` action, you can use the following rule:

```
url.scheme=ftp action.http_redirect(no)
```

Note: An error results if two `redirect( )` actions conflict. The error is noted at compile time if the conflicting actions are within the same action definition block. A runtime error is recorded in the event log if the conflicting actions are defined in different blocks.

Important: It is possible to put the browser into an infinite redirection loop if the URL that the browser is being redirected to also triggers a policy-based redirect response.

Syntax

```
redirect(response_code, regex_pattern, replacement_url)
```

where:

- *`response_code`*—An HTTP redirect code used as the HTTP response code; supported codes are 301, 302, 305, and 307.
- *`regex_pattern`*—A quoted regular-expression pattern that is compared with the request URL based on an anchored match. If the *`regex_pattern`* does not match the request URL, the redirect action is ignored. A *`regex_pattern`* match sets the values for substitution variables. If no variable substitution is performed by the *`replacement_url`* string, specify `".* "` for *`regex_pattern`* to match all request URLs. For more information about regular expressions, refer to Appendix E, "Using Regular Expressions," in the *Blue Coat* Proxy*SG Configuration and Management Guide*.
- *`replacement_url`*—A quoted string that can optionally include one or more variable substitutions, which replaces the entire URL once the substitutions are performed. The resulting URL is considered complete, and replaces any URL that contains a substring matching the *`regex_pattern`* substring. Sub-patterns of the *`regex_pattern`* matched can be substituted in *`replacement_url`* using the `$(`*`n`*`)` syntax, where *`n`* is an integer from 1 to 32, specifying the matched sub-pattern. For more information, see Appendix D: "CPL Substitutions".

Layer and Transaction Notes

- Use in `<Proxy>` or `<Cache>` layers.
- Do not use in `<Admin>`, `<Forward>`, or `<Exception>` layers.

See Also

- Actions: `rewrite(url.host, `*`host_regex_pattern`*`, `*`replacement_host`*`)`, `rewrite(url, `*`regex_pattern`*`, `*`replacement_url`*`)`, `set(url.port, `*`port_number`*`)`
- Conditions: `exception.id=`

replace( )

This deprecated action has been replaced by `rewrite( )`. For more information, see "rewrite( )" on page 237.

rewrite( )

Rewrites the request URL, URL host, or components of the specified header if it matches the regular-expression pattern. This action is often used in conjunction with the URL rewrite form of the transform action in a *server portal* application.

Note: The URL form of the `rewrite( )` action does not rewrite some URL components for Windows Media (MMS) transactions. The URL scheme, host, and port are restored to their original values and an error logged if the URL specified by *`replacement_url`* attempts to change these components.

An error results if the URL or URL host form of this action conflicts with another URL rewriting action. The error is noted at compile time if the conflicting actions are within the same action definition block. A runtime error is recorded in the event log if the conflicting actions are defined in different blocks.

Similarly, an error results if two header modifications act on the same header.

HTTPS Limitation

Only the host and port are available for rewriting by the URL or URL host form when the client browser is using a proxy for an HTTPS connection and the CONNECT or TUNNEL method is used. This is because the URL path is encrypted and unavailable for rewriting.

Syntax

```
rewrite(url, regex_pattern, replacement_url[, URL_form1, ...])
rewrite(url.host, regex_pattern, replacement_host[, URL_form1, ...])
rewrite(header, regex_pattern, replacement_component)
```

where:

- `url`—Specifies a rewrite of the entire URL.
- `url.host`—Specifies a rewrite of the host portion of the URL.
- *`header`*—Specifies the header to rewrite, using the following form. For a list of recognized headers, see Appendix C.
  - `request.header.`*`header_name`*—Identifies a recognized HTTP request header.
  - `response.header.`*`header_name`*—Identifies a recognized HTTP response header.
  - `request.x_header.`*`header_name`*—Identifies any request header, including custom headers.
  - `response.x_header.`*`header_name`*—Identifies any response header, including custom headers.
- *`regex_pattern`*—A quoted regular-expression pattern that is compared with the URL, host or header as specified, based on an anchored match. If the *`regex_pattern`* does not match, the rewrite action is ignored. A *`regex_pattern`* match sets the values for substitution variables. If the rewrite should always be applied, but no variable substitution is required for the replacement string, specify `".* "` for *`regex_pattern`*. For more information about regular expressions, refer to Appendix E, "Using Regular Expressions," in the *Blue Coat* Proxy*SG Configuration and Management Guide*.
- *`replacement_url`*—A quoted string that can optionally include one or more variable substitutions, which replaces the entire URL once the substitutions are performed. The resulting

URL is considered complete, and replaces any URL that contains a substring matching the *regex_pattern* substring. Sub-patterns of the *regex_pattern* matched can be substituted in *replacement_url* using the $(*n*) syntax, where *n* is an integer from 1 to 32, specifying the matched sub-pattern. For more information, see Appendix D: "CPL Substitutions".

- *replacement_host*—A quoted string that can optionally include one or more variable substitutions, which replaces the host portion of the URL once the substitutions are performed. Note that the resulting host is considered complete, and it replaces the host in the URL forms specified. Sub-patterns of the *regex_pattern* matched can be substituted in *replacement_host* using the $(*n*) syntax, where *n* is an integer from 1 to 32, specifying the matched sub-pattern. For more information, see Appendix D: "CPL Substitutions".
- *URL_form1, ...*—An optional list of up to three forms of the request URLs that will have the URL or host replaced. If this parameter is left blank, all three forms are rewritten. The following are the possible values:
  - log—Request URL used when generating log messages.
  - cache—Request URL used to address the object in the local cache .
  - server—Request URL sent to the origin server.
- *replacement_component*—A quoted string that can optionally include one or more variable substitutions, which replaces the entire component of the header matched by the *regex_pattern* substring. Sub-patterns of the *regex_pattern* matched can be substituted in *replacement_component* using the $(*n*) syntax, where *n* is an integer from 1 to 32, indicating the matched sub-pattern. For more information, see Appendix D: "CPL Substitutions".

Discussion

Any rewrite of the server form of the request URL must be respected by policy controlling upstream connections. The server form of the URL is tested by the server_url= conditions, which are the only URL tests allowed in <Forward> layers.

All forms of the URL are available for access logging. The version of the URL that appears in a specific access log is selected by including the appropriate substitution variable in the access log format:

- c-uri—The original URL
- cs-uri—The log URL, used when generating log messages
- s-uri—The cache URL, used to address the object in the local cache
- sr-uri—The server URL, used in the upstream request

In the absence of actions that modify the URL, all of these substitution variables represent the same value.

Layer and Transaction Notes

- Use in <Proxy> and <Cache> layers.
- Do not use in <Exception>, <Forward>, or <Admin> layers.
- URL and host rewrites apply to all transactions. Header rewrites apply to HTTP transactions.

Example

```
rewrite(url, "^http://www\.ijk\.com/(.*)", "http://www.server1.ijk.com/$(1)")
```

See Also

- Actions: `append( )`, `delete( )`, `delete_matching( )`, `redirect( )`, `set( )`, `transform`
- Conditions: `request.header.`*`header_name`*`=`, `request.header.`*`header_name`*`.address=`, `request.x_header.`*`header_name`*`=`, `request.x_header.`*`header_name`*`.address=`, `response.header.`*`header_name`*`=`, `response.x_header.`*`header_name`*`=`, `server_url=`
- Definitions: `transform url.rewrite`

set( )

Sets the specified header to the specified string after deleting all components of the header.

Note: An error results if two header modification actions modify the same header. The error is noted at compile time if the conflicting actions are within the same action definition block. A runtime error is recorded in the event log if the conflicting actions are defined in different blocks.

HTTPS Limitation

Only the host and port are available for setting when the client browser is using a proxy for an HTTPS connection and the CONNECT or TUNNEL method is used. This is because the URL path is encrypted and unavailable for setting.

Syntax

```
set(header, string)
set(im.message.text, value)
set(url.port, port_number [, URL_form1, URL_form2, ...])
```

where:

- `header`—A header specified using the following form. For a list of recognized headers, see Appendix C in this manual.
  - `request.header.`*`header_name`*—Identifies a recognized HTTP request header.
  - `response.header.`*`header_name`*—Identifies a recognized HTTP response header.
  - `request.x_header.`*`header_name`*—Identifies any request header, including custom headers.
  - `response.x_header.`*`header_name`*—Identifies any response header, including custom headers.
  - `exception.response.header.`*`header_name`*—Identifies a recognized HTTP response header from the exception response.
  - `exception.response.x_header.`*`header_name`*—Identifies any response header from the exception response, including custom headers.
- *`string`*—A quoted string that can optionally include one or more variable substitutions, which replaces the specified header components once the substitutions are performed.
- `im.message.text`, *`value`*—Sets the instant message text to the specified *`value`*.
- *`port_number`*—The port number that the request URL is set to. The range is an integer between 1 and 65535.
- *`URL_form1, URL_form2, ...`*—An optional list of up to three forms of the request URLs that have the port number set. If this parameter is left blank, all three forms of the request URL are rewritten. The possible values are the following:
  - `log`—Request URL used when generating log messages.
  - `cache`—Request URL used to address the object in the local cache.
  - `server`—Request URL sent to the origin server.

Discussion

Any change to the server form of the request URL must be respected by policy controlling upstream connections. The server form of the URL is tested by the `server_url=` conditions, which are the only URL tests allowed in `<Forward>` layers.

All forms of the URL are available for access logging. The version of the URL that appears in a specific access log is selected by including the appropriate substitution variable in the access log format:

- `c-uri`—The original URL.
- `cs-uri`—The log URL, used when generating log messages.
- `s-uri`—The cache URL, used to address the object in the local cache.
- `sr-uri`—The server URL, used in the upstream request.

In the absence of actions that modify the URL, all of these substitution variables represent the same value.

Layer and Transaction Notes

- Do not use in `<Admin>` or `<Forward>` layers.
- Use with `exception.response.header.`*`header_name`* in `<Proxy>` or `<Exception>` layers; otherwise use only from `<Proxy>` or `<Cache>` layers.
- When used with headers, applies to HTTP transactions.
- When used with `im.message.text`, applies to IM transactions.
- When used with `url.port`, applies to all transactions.

Example

```
; Modifies the URL port component to 8081 for requests sent to the server and cache.
set(url.port, 8081, server, cache)
```

See Also

- Actions: `append( )`, `delete( )`, `delete_matching( )`, `redirect( )`, `rewrite(url.host, regex_pattern, replacement_host)`, `rewrite(url,` *`regex_pattern`*`, replacement_url)`
- Conditions: `request.header.`*`header_name=`*`, request.header.`*`header_name`*`.address=, request.x_header.`*`header_name=`*`, request.x_header.`*`header_name`*`.address=, response.header.`*`header_name=`*`, response.x_header.`*`header_name=`*`, server_url=`

transform

Invokes an active content or URL rewrite transformer. The invoked transformer takes effect only if the `transform` action is used in a define action definition block, and that block is in turn enabled by an `action( )` property.

See chapters 11 and 13 in the *Configuration and Management Guide* for examples of how this action is used with the active content and URL rewrite transformers.

Note: Any transformed content is not cached, in contrast with content that has been sent to a virus scanning server. This means the transform action can be safely triggered based on any condition, including client identity and time of day.

Syntax

```
transform transformer_id
```

where *`transformer_id`* is a user-defined identifier for a transformer definition block. This identifier is not case-sensitive.

Layer and Transaction Notes

- Use in `<Proxy>` or `<Cache>` layers.
- Do not use in `<Admin>`, `<Forward>`, or `<Exception>` layers.

Example

```
; The transform action is part of an action block enabled by a rule.

<proxy>
    url.domain=!my_site.com action.strip_active_content(yes)

; transformer definition

define active_content strip_with_indication

tag_replace applet <<EOT
   <B>APPLET content has been removed</B>
EOT

tag_replace embed <<EOT
   <B>APPLET content has been removed</B>
EOT

tag_replace object <<EOT
   <B>OBJECT content has been removed</B>
EOT

tag_replace script <<EOT
   <B>SCRIPT content has been removed</B>
EOT
end

define action strip_active_content
    ; the transform action invokes the transformer
    transform strip_with_indication
end
```

See Also

- Properties: `action( )`
- Definitions: `define action`, `transform active_content`, `transform url.rewrite`

virus_check( )

This deprecated action sends the requested document to a virus scanning server. For more information, see "response.icap_service( )" on page 213.

Chapter 6: *Definition Reference*

In policy files, definitions serve to bind a set of conditions, actions, or transformations to a user-defined label.

Two types of definitions exist:

- Named definitions—Explicitly referenced by policy.
- Anonymous definitions—Apply to all policy evaluation and are not referenced directly in rules.

There are two types of anonymous definitions: DNS and RDNS restrictions.

Definition Names

There are various types of named definitions. Each of these definitions is given a user-defined name that is then used in rules to refer to the definitions. The user-defined labels used with definitions are not case-sensitive. Characters in labels may include the following:

- letters
- numbers
- space
- period
- underscore
- hyphen
- forward slash
- ampersand

The first character of the name must be a letter or underscore. If spaces are included, the name must be a quoted string.

Only alphanumeric, underscore, and dash characters can be used in the name given to a defined action.

The remainder of this chapter lists the definitions and their accepted values. It also provides tips as to where each definition can be used and examples of how to use them.

define action

Binds a user-defined label to a sequence of action statements. The `action( )` property has syntax that allows for individual action definition blocks to be enabled and disabled independently, based on the policy evaluation for the transaction. When an action definition block is enabled, any action statements it contains operate on the transaction as indicated by their respective arguments. See Chapter 5: "Action Reference" for more information about the various action statements available.

Note: Action statements that must be performed in a set sequence and cannot overlap are best listed within a single action definition block

Syntax

```
define action label
    list of action statements
end [action label]
```

where:

- *`label`*—A user-defined identifier for an action definition. Only alphanumeric, underscore, and dash characters can be used in the label given to a defined action.
- *`list of action statements`*—A list of actions to be carried out in sequence. See Chapter 5 Action Reference for the available actions.

Layer and Transaction Notes

Each action statement has its own timing requirements and layer applicability. The timing requirements for the overall action are the strictest required by any of the action statements contained in the definition block.

Similarly, the layers that can reference an action definition block are the layers common to all the action statements in the block.

Action statements that are not appropriate to the transaction will be ignored.

Example

The following is a sample action given the name `scrub_private_info`, that clears the `From` and `Referer` headers (which normally could be used to identify the user and where they clicked from) in any request going to servers not in the internal domain.

```
<cache>
    url.domain=!my_internal_site.com action.scrub_private_info(yes)

define action scrub_private_info
    set( request.header.From, "" )
    set( request.header.Referer, "" )
end
```

Notice that the object on which the `set( )` action operates is given in the first argument, and then appropriate values follow, in this case, the new value for the specified header. This is common to many of the actions.

See Also

- Properties: `action( )`

- Definitions: `transform active_content`, `transform url_rewrite`
- Chapter 5: "Action Reference".

define active_content

Defines rules for removing or replacing active content in HTML or ASX documents. This definition takes effect only if it is invoked by a `transform` action in a `define action` definition block, and that block is in turn enabled an `action( )` property as a result of policy evaluation.

Active content transformation acts on the following four HTML elements in documents: `<applet>`, `<embed>`, `<object>`, and `<script>`. In addition, a script transformation removes any JavaScript content on the page. For each tag, the replacement can either be empty (thus deleting the tag and its content) or new text that replaces the tag. Multiple tags can be transformed in a single active content transformer. Pages served over an HTTPS tunneled connection are encrypted so the content cannot be modified.

Note: Transformed content is not cached, in contrast with content that has been sent to a virus scanning server. Therefore, a transformer can be safely triggered based on any condition, including client identity and time of day.

Replaces: `transform active_content`

Syntax

```
define active_content transformer_id
  tag_replace HTML_tag_name << text_end_delimiter
  [replacement_text]
  text_end_delimiter
  [tag_replace ...]
  ...
end
```

where:

- *`transformer_id`*—A user-defined identifier for a transformer definition block. Used to invoke the transformer using the `transform` action in a `define action` definition block.
- *`HTML_tag_name`*—The name of an HTML tag to be removed or replaced, as follows:
  - `applet`—Operates on the `<applet>` element, which places a Java applet on a web page.
  - `embed`—Operates on the `<embed>` element, which embeds an object, such as a multimedia file, on a web page.
  - `object`—Operates on the `<object>` element, which places an object, such as an applet or media file, on a web page.
  - `script`—Operates on the `<script>` element, which adds a script to a web page. Also removes any JavaScript entities, strings, or events that may appear on the page.

  If the `tag_replace` keyword is repeated within the body of the transformer, multiple HTML tags can be removed or replaced.
- *`text_end_delimiter`*—A user-defined token that does not appear in the replacement text and does not use quotes or whitespace. The delimiter is defined on the first line, after the required double angle brackets (<<). All text that follows, up to the second use of the delimiter, is used as the replacement text.
- *`replacement_text`*—Either blank, to remove the specified tag, or new text (including HTML tags) to replace the tag.

Layer and Transaction Notes

- Applies to proxy transactions.
- Only alphanumeric, underscore, dash, and slash characters can be used with the define action name.

Example

```
<proxy>

url.domain=!my_site.com action.strip_active_content(yes)

define active_content strip_with_indication
    tag_replace applet <<EOT
            <B>APPLET content has been removed</B>
    EOT
    tag_replace embed <<EOT
            <B>APPLET content has been removed</B>
    EOT
    tag_replace object <<EOT
            <B>OBJECT content has been removed</B>
    EOT
    tag_replace script <<EOT
            <B>SCRIPT content has been removed</B>
    EOT
end

define action strip_active_content
    transform strip_with_indication
end
```

See Also

- Actions: `transform`
- Definitions: `define action`, `define url.rewrite`
- Properties: `action( )`

define category

Category definitions are used to extend vendor content categories or to create your own. The *category_name* definition can be used anywhere a content filter category name would normally be used, including in `category=` tests.

Definitions can include other definitions to create a hierarchy. For example, sports could include football by including `category=football` in the definition for sports. A defined category can have at most one parent category (multiple inheritance is not allowed).

Multiple definitions using the same *category_name* are coalesced together.

When policy tests a request URL to determine if it is in one of the categories specified by a trigger, all sub-categories are also checked (see Examples).

Syntax

```
define category category_name
    urlpaths
end [category_name]
```

where:

- *category_name*—If *category_name* matches the name of an existing category from the configured content filtering service, this is used to extend the coverage of that category; otherwise it defines a new user defined category. *category_name* can be used anywhere a content filter category name would normally be used, including in `category=` tests.
- *urlpaths*—A list of domain suffix or path prefix expressions, as used in the `url.domain=` condition.You only need to specify a partial URL:
  - Hosts and subdomains within the domain you specify are automatically included.
  - If you specify a path, all paths with that prefix will be included (if you specify no path, the entire site is included).

Layer and Transaction Notes

- Use in `<Proxy>` and `<Cache>` Layers.
- Applies to all transactions.

Examples

The following example illustrates some of the variations allowed in a category definition:

```
define category Grand_Canyon
    kaibab.org
    www2.nature.nps.gov/ard/parks/grca/
    nps.gov/grca/
    grandcanyon.org
end
```

The following definitions define the categories sports and football, and make football a sub-category of sports:

```
define category sports
    sports.com
```

```
    sportsworld.com
    category=football ; include subcategory
end

define category football
    nfl.com
    cfl.ca
end
```

The following policy needs only to refer to the sports category to also test the sub-category football:

```
<Proxy>
    deny category=sports ; includes subcategories
```

For more information on using `category=` tests, including examples, refer to Chapter 17: "Content Filtering," in the Proxy*SG Configuration and Management Guide*.

See Also

- Conditions: `category=`
- Properties: `action( )`

define condition

Binds a user-defined label to a set of conditions for use in a `condition=` expression.

For condition definitions, the manner in which the condition expressions are listed is significant. Multiple condition expressions on one line, separated by whitespace, are considered to have a Boolean AND relationship. However, the lines of condition expressions are considered to have a Boolean OR relationship.

Performance optimized condition definitions are available for testing large numbers of URLs. See `define url condition`, `define url.domain condition`, and `define server_url.domain condition`.

Syntax

```
define condition label
    condition_expression ...

    ...
end [condition labe]
```

where:

- *`label`*—A user-defined identifier for a condition definition. Used to call the definition from an `action.action_label( )` property.
- *`condition_expression`*—Any of the conditions available in a rule. The layer and timing restrictions for the defined condition depend on the layer and timing restrictions of the contained expressions.

The `condition=`*`condition`* is one of the expressions that can be included in the body of a `define condition` definition block. In this way, one condition definition block can call another condition-related definition block, so that they are in effect *nested*. Circular references generate a compile error.

Layer and Transaction Notes

The layers that can reference a condition definition are the layers common to all the condition statements in the block.

A condition can be evaluated for any transaction. The condition evaluates to true if all the condition expressions on any line of the condition definition apply to that transaction and evaluate to true. Condition expressions that do not apply to the transaction evaluate to false.

Example

This example illustrates a simple virus scanning policy designed to prevent some traffic from going to the scanner. Some file types are assumed to be at low risk of infection (some virus scanners will not scan certain file types), and some are assumed to have already been scanned when they were loaded on the company's servers.

Note: The following policy is not a security recommendation, but an illustration of a technique. If you choose to selectively direct traffic to your virus scanner, you should make your own security risk assessments based on current information and knowledge of your virus scanning vendor's capabilities.

```
define condition extension_low_risk ; file types assumed to be low risk.
   url.extension=(asf,asx,gif,jpeg,mov,mp3,ram,rm,smi,smil,swf,txt,wax,wma,wmv,wvx)
end

define condition internal_prescanned ; will be prescanned so we can assume safe
   server_url.domain=internal.myco.com server_url.extension=(doc,dot,hlp,html)
   server_url.domain=internal.myco.com \
       response_header.Content-Type=(text, application/pdf)
end

define condition white_list
   condition=extension_low_risk
   condition=internal_prescanned
end

<cache>
   condition=!internal_white_list action.virus_scan(true)

define action virus_scan
   response.icap_service( "ICAP_server" ) ; configured service name
end
```

See Also

- Conditions: `category=`, `condition=`
- Properties: `action.`*`action_label`*`( )`

define domain

This deprecated syntax has been replaced by the `url.domain` condition. For more information see "define url.domain condition" on page 263.

define javascript

A javascript definition is used to define a *javascript transformer,* which adds javascript that you supply to HTML responses.

Syntax

```
define javascript transformer_id
    javascript-statement
    [javascript-statement]
    …
end
```

where:

- *transformer_id*—A user-defined identifier for a transformer definition block. Used to invoke the transformer using the `transform` action in a `define action` definition block.
- A *javascript-statement* has the following syntax:

  javascript-statement ::= *section-type replacement*
  section-type ::= prolog | onload | epilog
  replacement ::= << *endmarker newline lines-of-text newline endmarker*

This allows you to specify a block of javascript to be inserted at the beginning of the HTML page (prolog), to be inserted at the end of the HMTL page (epilog), and to be executed when parsing is complete and the page is loaded (onload). Each of the section types is optional.

Layer and Transaction Notes

Applies to proxy transactions.

Example

The following is an example of a javascript transformer that adds a message to the top of each Web page, used as part of a simple content filtering application:

```
define javascript js_transformer
  onload <<EOS
    var msg = "This site is restricted. Your access has been logged.";
    var p = document.createElement("p");
    p.appendChild(document.createTextNode(msg));
    document.body.insertBefore(p, document.body.firstChild);
  EOS
end

define action js_action
    transform js_transformer
end

<proxy>
    category=restricted action.js_action(yes)
```

The VPM uses javascript transformers to implement popup ad blocking.

See Also

- Actions: `transform`
- Definitions: `define action`
- Properties: `action( )`

define prefix condition

This deprecated syntax has been replaced by the define url condition. For more information see "define url condition" on page 261.

define server_url.domain condition

Binds a user-defined label to a set of domain-suffix patterns for use in a `condition=` expression. Using this definition block allows you to quickly test a large set of `server_url.domain=` conditions. Although the `define condition` definition block could be used in a similar way to encapsulate a set of domain suffix patterns, this specialized definition block provides a substantial performance boost.

The manner in which the URL patterns and any condition expressions are listed is significant. Each line begins with a URL pattern and, optionally, one or more condition expressions, all of which have a Boolean AND relationship. Each line inside the definition block is considered to have a Boolean OR relationship with other lines in the block.

For more information about choosing the best way to test a request URL, see "Denying Access to URLs" in Chapter 9 of the *Configuration and Management Guide*.

Note: This condition is for use in the `<Forward>` layers and takes into account the effect of any `rewrite( )` actions on the URL. Because any rewrites of the URL intended for servers or other upstream devices must be respected by `<Forward>` layer policy, conditions that test the unrewritten URL are not allowed in `<Forward>` layers. Instead, this condition is provided.

Syntax

```
define server_url.domain condition label
    domain_suffix_pattern [condition_expression ...]
    ...
end [server_url.domain condition label]
```

where:

- *`label`*—A user-defined identifier for a domain condition definition. Used in a `condition=` condition.
- *`domain_suffix_pattern`*—A URL pattern that includes a domain name (domain), as a minimum. See the `url=` condition reference for a complete description.
- *`condition_expression`* ...—An optional condition expression, using any of the conditions available in a rule, that are allowed in a `<Forward>` layer. For more information, see Chapter 3: "Condition Reference".

The `condition=` condition is one of the expressions that can be included in the body of a `define server_url.domain condition` definition block, following a URL pattern. In this way, one `server_url.domain` definition block can call another condition-related definition block, so that they are in effect *nested*. See the example in the `define condition` definition block topic. Any referenced condition must be valid in a `<Forward>` layer.

Layer and Transaction Notes

- Use in `<Forward>` layers.
- Applies to all transactions.

Example

```
define server_url.domain condition allowed
    inventory.example.com
```

```
    affinityclub.example.com
end

<Forward>
    condition=!allowed access_server(no)
```

See Also

Condition: `condition=`, `server_url.domain=`

Definitions: `define url.domain condition`

define subnet

Binds a user-defined label to a set of IP addresses or IP subnet patterns. Use a subnet definition label with any of the conditions that test part of the transaction as an IP address, including: `client.address=`, `proxy.address=`, `request.header.`*`header_name`*`.address=`, `request.x_header.`*`header_name`*`.address`, and `server_url.address=`.

The listed IP addresses or subnets are considered to have a Boolean OR relationship, no matter whether they are all on one line or separate lines.

Syntax

```
define subnet label
    { ip_address | subnet } { ip_address | subnet } ...
    ...
end [subnet label]
```

where:

- *`label`*—A user-defined identifier for this subnet definition.
- *`ip_address`*—IP address; for example, `10.1.198.0`.
- *`subnet`*—Subnet specification; for example, `10.25.198.0/16`.

Example

```
define subnet local_net
    1.2.3.4  1.2.3.5 ; can list individual IP addresses
    2.3.4.0/24  2.3.5.0/24  ; or subnets
end
```

```
<proxy>
  client.address=!local_subnet deny
```

See Also

- Conditions: `client.address=`, `proxy.address=`, `request.header.`*`header_name`*`.address=`, `request.x_header.`*`header_name`*`.address`, and `server_url.address=`

define url condition

Binds a user-defined label to a set of URL prefix patterns for use in a `condition=` expression. Using this definition block allows you to quickly test a large set of `url=` conditions. Although the `define condition` definition block could be used in a similar way to encapsulate a set of URL prefix patterns, this specialized definition block provides a substantial performance boost.

The manner in which the URL patterns and any condition expressions are listed is significant. Each line begins with a URL pattern suitable to a `url=` condition and, optionally, one or more condition expressions, all of which have a Boolean AND relationship. Each line inside the definition block is considered to have a Boolean OR relationship with other lines in the block.

Syntax

```
define url condition label
    url_prefix_pattern [condition_expression ...]
    ...
end [url condition label]
```

where:

- *`label`*—A user-defined identifier for a prefix condition definition.
- *`url_prefix_pattern`* ... —A URL pattern that includes at least a portion of the following:

  `scheme://host:port/path`

  - *`scheme`*—A URL scheme (`http`, `https`, `ftp`, `mms`, or `rtsp`) followed by a colon (`:`).
  - *`host`*—A host name or IP address, optionally preceded by two forward slashes (`//`). Host names must be complete; for example, `url=http://www` will fail to match a URL such as `http://www.example.com`. This use of a complete host instead of simply a domain name (such as `example.com`) marks the difference between the prefix and domain condition definition blocks.
  - *`port`*—A port number, between 1 and 65535.
  - *`path`*—A forward slash (`/`) followed by one or more full directory names.

  Accepted prefix patterns include the following:

  ```
  scheme://host
  scheme://host:port
  scheme://host:port/path
  scheme://host/path
  //host
  //host:port
  //host:port/path
  //host/path
  host
  host:port
  host:port/path
  host/path
  /path
  ```

- *`condition_expression`* ...—An optional condition expression, using any of the conditions available in a rule. For more information, see Chapter 3: "Condition Reference". The layer and

timing restrictions for the defined condition will depend on the layer and timing restrictions of the contained expressions.

The `condition=` condition is one of the expressions that can be included in the body of a `define url condition` definition block, following a URL pattern. In this way, one prefix definition block can call another condition-related definition block, so that they are in effect *nested*. See the example in the `define condition` definition block topic.

Example

```
define url condition allowed
    http://www.inventory.example.com method=GET
    www.affinityclub.example.com/public ; any scheme allowed
end

<proxy>
    condition=allowed allow
```

See Also

Conditions: `category=`, `condition=`, `url=`

Definitions: `define url.domain condition`

define url.domain condition

Binds a user-defined label to a set of domain-suffix patterns for use in a `condition=` expression. Using this definition block allows you to test a large set of `server_url.domain=` conditions very quickly. Although the `define condition` definition block could be used in a similar way to encapsulate a set of domain suffix patterns, this specialized definition block provides a substantial performance boost.

For domain and prefix definitions, the manner in which the URL patterns and any condition expressions are listed is significant. Each line begins with a URL pattern and, optionally, one or more condition expressions, all of which have a Boolean AND relationship. Each line inside the definition block is considered to have a Boolean OR relationship with other lines in the block.

Syntax

```
define url.domain condition label
    domain_suffix_pattern [condition_expression ...]
    ...
end [url.domain condition label]
```

where:

- *`label`*—A user-defined identifier for a domain condition definition. Used in a `condition=` condition.
- *`domain_suffix_pattern`*—A URL pattern suitable to the `url.domain=` condition, that includes a domain name (domain), as a minimum. See the `url=` condition reference for a complete description.
- *`condition_expression`* ...—An optional condition expression, using any of the conditions available in a rule. For more information, see Chapter 3: "Condition Reference". The layer and timing restrictions for the defined condition will depend on the layer and timing restrictions of the contained expressions.

The `condition=` condition is one of the expressions that can be included in the body of a `define url.domain condition` definition block, following a URL pattern. In this way, one domain definition block can call another condition-related definition block, so that they are in effect *nested*. See the example in the `define condition` definition block topic.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, and `<Exception>` layers.
- Applies to all transactions.

Example

```
define domain condition allowed
    inventory.example.com method=GET
    affinityclub.example.com
end

<proxy>
    condition=allowed allow
```

See Also

- Condition: `condition=`, `server_url.domain=`
- Definitions: `define url condition`, `define server_url.domain condition`

define url_rewrite

Defines rules for rewriting URLs embedded in tags within HTML, CSS, JavaScript or ASX documents. This transformer takes effect only if it is also invoked by a `transform` action in a `define action` definition block, and that block is in turn called from an `action( )` property.

For each url found within an HTTP response, a `url_rewrite` transformer first converts the URL into absolute form, then finds the first `subst_embedded` or `subst_prefix` statement whose *`server_URL_substring`* matches the URL being considered. If such a match is found, then that substring is replaced by the *`client_url_substring`*.

Matching is case-sensitive by default; use the optional `caseless` keyword for case-insensitive matching. All `subst_embedded` statements after the first occurrence of the caseless keyword use case-insensitive string matching.

Multiple URL prefix substitutions can be made in a single `define url_rewrite` definition block. This type of transformation is often used in conjunction with the request URL form of the `rewrite( )` action in a *server portal* application.

To find URLs within an HTTP response, the Proxy*SG* looks for `Location:`, `Content-Location:`, and `Refresh:` headers, and parses HTML, JavaScript, CSS, and ASX files. The Proxy*SG* does not search for, nor rewrite relative URLs embedded within Javascript.

Note: Pages served over an HTTPS tunneled connection are encrypted, so URLs embedded within them cannot be rewritten.

Transformed content is not cached (although the original object may be), in contrast with content that has been sent to a virus scanning server. This means any transformer can be safely triggered based on any condition, including client identity and time of day.

Replaces: `transform url_rewrite`

Syntax

```
define url_rewrite transformer_id
  [caseless]
  subst_embedded "client_url_substring" "server_url_substring"
  subst_prefix "client_url_substring" "server_url_substring"
  ...
end
```

where:

- *`transformer_id`*—A user-defined identifier for a transformer definition block. Used to invoke the transformer using the `transform` action in a `define action` definition block.
- `subst_embedded`—Matches *`server_url_substring`* anywhere in the URL.
- `subst_prefix`—Matches *`server_url_substring`* as a prefix of the URL.
- *`client_url_substring`*—A string that will replace *`server_url_substring`* when that string is matched for a URL in the retrieved document. The portion of the URL that is not substituted is unchanged.

- *server_url_substring*—A string that, if found in the server URL, will be replaced by the *client_url_substring*. The comparison is done against original normalized URLs embedded in the document.

Note: Both client_url_substring and server_url_substring are literal strings. Wildcard characters and regular expression patterns are not supported.

Discussion

If there are a series of subst_embedded and subst_prefix statements in a url_rewrite definition, the first statement to match a URL takes effect and terminates processing for that URL.

Layer and Transaction Notes

Applies to proxy transactions.

Example

```
<Proxy> ; server portal for IJK
  url=ijk.com/ action.ijk_server_portal(yes)

; This transformation provides server portaling for IJK non video content

define url_rewrite ijk_portal
  caseless
  subst_embedded "http://www.ijk.com/" "http://www.server1.ijk.com/"
end

; This action runs the transform for IJK server portaling for http content
; Note that the action is responsible for rewriting related headers

define action ijk_server_portal
  ; request rewriting
  rewrite( url, "^http://www\.ijk\.com/(.*)", "http://www.server1.ijk.com/`(1)" )
  rewrite( request.header.Referer, "^http://www\.ijk\.com/(.*)",
                                   "http://www.server1.ijk.com/`(1)" )
  ; response rewriting
  transform ijk_portal
  rewrite( response.header.Location, "^http://www\.server1\.ijk\.com/(.*)",
                                     "http://www.ijk.com/`(1)" )
end
```

See Also

- Actions: transform
- Definitions: define action, define active_content
- Properties: action( )

restrict dns

This definition restricts DNS lookups and is useful in installations where access to DNS resolution is limited or problematic. The definition has no name because it is not directly referenced by any rules. It is global to policy evaluation and intended to prevent any DNS lookups caused by policy. It does not suppress DNS lookups that might be required to make upstream connections.

If the domain specified in a URL matches any of the domain patterns specified in `domain_list`, no DNS lookup is done for any `category=`, `url=`, `url.address=`, `url.domain=`, or `url.host=` test.

The special domain `"."` matches all domains, and therefore can be used to restrict all policy-based DNS lookups.

If a lookup is required to evaluate the trigger, the trigger evaluates to false.

A `restrict dns` definition may appear multiple times in policy. The compiler attempts to coalesce these definitions, and may emit various errors or warnings while coalescing if the definition is contradictory or redundant.

Syntax

```
restrict dns
    restricted_domain_list
  except
    exempted_domain_list
end
```

where

- *`restricted_domain_list`*—Domains for which DNS lookup is restricted.
- *`exempted_domain_list`*—Domains exempt from the DNS restriction. Policy is able to use DNS lookups when evaluating policy related to these domains.

Layer and Transaction Notes

Applies to all layers and transactions.

Example

The following definition restricts DNS resolution to all but `mydomain.com`:

```
restrict dns
    . ; meaning “all”
  except
    mydomain.com
end
```

See Also

- Conditions: `category=`, `url=`, `server_url=`
- Definitions: `restrict rdns`

restrict rdns

This definition restricts reverse DNS lookups and is useful in installations where access to reverse DNS resolution is limited or problematic. The definition has no name. It is global to policy evaluation and is not directly referenced by any rules.

If the requested URL specifies the host in IP form, no reverse DNS lookup is performed to match any `category=`, `url=`, `url.domain=`, or `url.host=` condition.

The special token *all* matches all subnets, and therefore can be used to restrict all policy-based reverse DNS lookups.

If a lookup is required to evaluate the trigger, the trigger evaluates to false.

A `restrict rdns` definition may appear multiple times in policy. The compiler attempts to coalesce these definitions, and may emit various errors or warnings while coalescing if the definition is contradictory or redundant.

Syntax

```
restrict rdns
    restricted_subnet_list
  except
    exempted_subnet_list
end
```

where

- *`restricted_subnet_list`*—Subnets for which reverse DNS lookup is restricted.
- *`exempted_subnet_list`*—Subnets exempt from the reverse DNS restriction. Policy is able to use reverse DNS lookups when evaluating policy related to these subnets.

Layer and Transaction Notes

Applies to all layers and transactions.

Example

The following definition restricts reverse DNS resolution for all but the 10.10.100.0/24 subnet:

```
restrict rdns
    all
  except
    10.10.100.0/24
end
```

See Also

- Conditions: `category=`, `url=`, `server_url=`
- Definitions: `restrict dns`

transform active_content

This deprecated syntax has been replaced by `define active_content`. For more information see "define active_content" on page 248.

transform url_rewrite

This deprecated syntax has been replaced by `define url_rewrite`. For more information see "define url_rewrite" on page 265.

Appendix A: *Glossary*

actions	A class of definitions. CPL has two general classes of actions: request or response modifications and notifications. An action takes arguments (such as the portion of the request or response to modify) and is wrapped in a named action definition block. When the action definition is turned on by the policy rules, any actions it contains operate on their respective arguments.
`<Admin>` layer	One of the five layer types allowed in a policy. Used to define policy rules that control access to the Management Console and command line interface (CLI).
admin transaction	Encapsulation of a request to manage the Proxy*SG* for the purposes of policy evaluation. Policy in `<Admin>` layers applies to admin transactions. Additionally, if the user is explicitly proxied to the Proxy*SG*, a proxy transaction will also be created for the request.
allow	The preferred short form of `exception(no)`, a property setting that indicates that the request should be granted. A default rule for the proxy policy layer. You have two choices: allow or deny. Deny prevents any access to the Proxy*SG*; allow permits full access to the Proxy*SG*.
`<Cache>` layer	One of the five layer types allowed in a policy. Used to list policy rules that are evaluated during a cache or proxy transaction.
cache transaction	Encapsulation of a request, generated by the Proxy*SG* and directed at an upstream device, for the purposes of maintaining content in the local object store.
Central Policy File	A file provided by Blue Coat Technical Support to ensure that the Proxy*SG* behaves correctly and efficiently when accessing certain sites. You can adapt this file to include policies you want to share among multiple appliances.
condition	A boolean combination of trigger expressions that yields true or false when evaluated.
default policy	The default settings for various transaction properties taken from configuration. An important example is the default proxy policy that is configurable to either allow or deny
definition	A definition binds a user-defined label to a condition, a content category, a transformation or a group of actions.
deny	The preferred short form of `exception(policy_denied)`, a property setting that indicates that the request should be refused.
Evaluation order	The order in which the four policy files—Central, Local, VPM, and Forward—are evaluated. When a file is evaluated last, the policy rules and the related configuration settings it specifies can override any settings triggered in the other files. The order of evaluation of the Central, Local, and VPM policy files is configurable using the `policy order` CLI command or the Management Console. The Forward file is always last in the evaluation order.
Exception layer	One of the five layer types allowed in a policy. Exception layers are evaluated when an exception property is set, forcing transaction termination. Policy in an exception layer gives the administrator a final chance to modify the properties (such as headers) of the response (exception) object, just as they would get a chance to modify the properties of an object returned from the origin server or from cache.
`<Forward>` layer	One of the five layer types allowed in a policy. `<Forward>` layers are only evaluated when the current transaction requires an upstream connection.

Forward Policy File	A file you create or that might be created during an upgrade from prior SGOS versions, and that you maintain to supplement any policy described in the other three policy files. It is normally used for forwarding policy. The Forward policy file is always last in the evaluation order. Forwarding policy is generally distinct and independent of other policies, and is often used as part of maintaining network topologies. Forwarding policy can also be created and maintained through the Visual Policy Manager.
layer	A CPL construct for expressing the rules for a single policy decision. Multiple layers can be used to make multiple decisions. Layers are evaluated in top to bottom order. Decisions made by later layers can override decisions made by earlier layers. Layer evaluation terminates on the first rule match. Five layer types exist. The layer type defines the transactions evaluated against this policy and restricts the triggers and properties allowed in the rules used in the layer. Each of the five types of layers are allowed in any policy file.
Local Policy File	A file you create and maintain on your network for policy specific to one or more Proxy*SG* appliances. This is the file you would normally create when writing CPL directly with a text editor, for use on some subset of the Proxy*SG* appliances in your organization. On upgrade from a CacheOS 4.x system, the local file will contain any filter rules configured under the old system.
Match	When a rule is evaluated, if all triggers evaluate to true, then all properties specified are set. This is often referred to as a rule Match (for example in policy tracing.)
Miss	When a rule is evaluated, if any trigger evaluates to false, all properties specified are ignored. This is often referred to as a rule Miss (for example in policy tracing.)
N/A	The rule can't be evaluated for this transaction and is being skipped. N/A happens, for example, when you try to apply a streaming condition to an FTP transaction.
policy files	Any one of four files that contain CPL: Central, Local, VPM, or Forward. When the policy is installed, the contents of each of the files is concatenated according to the evaluation order.
policy trace	A listing of the results of policy evaluation. Policy tracing is useful when troubleshooting policy.
property	A CPL setting that controls some aspect of transaction processing according to its value. CPL properties have the form `property(setting)`. At the beginning of a transaction, all properties are set to their default values, many of which come from the configuration settings.
`<Proxy>` layer	One of the five layer types allowed in a policy, used to list policy rules that control access to proxy services configured on the Proxy*SG*. Rules in the `<Proxy>` layer include user authentication and authorization requirements, time of day restrictions, and content filtering.
proxy transaction	A transaction created for each request received over the proxy service ports configured on the Proxy*SG*. The proxy transaction covers both the request and its associated response, whether fetched from the origin server or the local object store.
request transformation	A modification of the request for an object (either the URL or Headers). This modification might result in fetching a different object, or fetching the object through a different mechanism.

response transformation	a modification of the object being returned. This modification can be to either the protocol headers associated with the response sent to the client, or a transformation of the object contents itself, such as the removal of active content from HTML pages.
rule	A list of triggers and property settings, written in any order. A rule can be written on multiple lines using a line continuation character. If the rule matches (all triggers evaluate to true), all properties will be set as specified. At most one rule per layer will match. Layer evaluation terminates on the first rule match.
section	A way of grouping rules of like syntax together. Sections consist of a section header that defines the section type, followed by policy rules.The section type determines the allowed syntax of the rules, and an evaluation strategy.
transaction	An encapsulation of a request to the Proxy*SG* together with the resulting response that can be subjected to policy evaluation. The version of policy current when the transaction starts is used for evaluation of the complete transaction, to ensure consistent results.
trigger	A named test of some aspect of a transaction. CPL triggers have the form *`trigger_name=value`*. Triggers are used in rules, and in condition definitions.
Visual Policy Manager file	A file created and stored on an individual Proxy*SG* by the Visual Policy Manager. The VPM allows you to create policies without writing CPL directly. Since the VPM supports a subset of CPL functionality, you might want to supplement any policy in a VPM file with rules in the Local policy file. If you have a new Proxy*SG*, the VPM file is empty. VPM files can be shared among various Proxy*SG* appliances by copying the VPM files to a Web server and then using the Management Console or the CLI from another Proxy*SG* to download and install the files.

Appendix B: *Testing and Troubleshooting*

If you are experiencing problems with your policy files or would like to monitor evaluation for brief periods of time, consider using the policy tracing capabilities of the policy language.

Tracing allows you to examine how the Proxy*SG* policy is applied to a particular request. To configure tracing in a policy file, you use several policy language properties to enable tracing, set the verbosity level, and specify the path for output. Using appropriate conditions to guard the tracing rules, you can be specific about the requests for which you gather tracing information.

Note: Use policy tracing for troubleshooting only. Tracing is best used temporarily for troubleshooting, while the `log_message( )` action is best for on-going monitoring. For more information about the `log_message( )` action, see "log_message( )" on page 232. If tracing is enabled in a production setting, Proxy*SG* performance degrades. After you complete troubleshooting, be sure to remove policy tracing.

CPL provides the following trace-related properties:

- `trace.rules( )`—Controls the tracing of rule evaluation. Trace can show which rules missed, which matched, and which were not applicable (N/A), meaning the rule cannot be evaluated for this transaction and is being skipped. N/A occurs, for example, when you try to apply a streaming trigger to an FTP transaction.
- `trace.request( )`—Enables tracing and includes a description of the transaction being processed in the trace. No trace output is generated if this is set to `no`.
- `trace.destination( )`—Directs the trace output to a user-named trace log.

Enabling Rule Tracing

Use the `trace.rules( )` property to enable or disable rule tracing. Rule tracing shows you which rules are executed during policy evaluation. This property uses the following syntax:

```
trace.rules(yes|no|all)
```

where

- `yes` enables rule tracing but shows matching rules only.
- `no` disables rule tracing.
- `all` enables tracing, with added detail about conditions that failed to match.

Example

The following enables tracing:

```
<proxy>
   trace.rules(yes) tracewhere:request(yes)
```

Enabling Request Tracing

Use the `trace.request( )` property to enable request tracing. Request tracing logs a summary of information about the transaction: request parameters, property settings, and the effects of all actions taken. This property uses the following syntax:

```
trace.request(yes|no)
```

where:

- `yes`—Includes request parameters, property settings, and the effects of all actions taken.
- `no`—Produces no tracing information, even if `trace.rules( )` is set.

Example

The following enables full tracing information for all transactions:

```
<cache>
     trace.rules(all) trace.request(yes)
```

Configuring the Path

Use the `trace.destination( )` property to configure where the ProxySG saves trace information. The trace destination can be set and reset repeatedly. It takes effect (and the trace is actually written) only when the ProxySG has finished processing the request and any associated response. Trace output is saved to an object that is accessible using a console URL in the following form:

```
https://ProxySG_IP_address:8081/Policy/Trace/path
```

where *path* is, by default, `default.trace.html`. This property allows you to change the destination. The property uses the following syntax:

```
trace.destination(path)
```

where *path* is a filename, directory path, or both. If you specify only a directory, the default trace filename is used.

You can view policy statistics through the Management Console: Statistics>Advanced>Policy>List of policy URLs.

Example

In the following example, two destinations are configured for policy tracing information:

```
<proxy>
   client_address=10.25.0.0/16 trace.destination(internal_trace.html)
   client_address=10.0.0.0/8 trace.destination(external_trace.html)
```

The console URLs for retrieving the information would be

```
http://<ProxySG_IP_address>:8081/Policy/Trace/internal_trace.html
http://<ProxySG_IP_address>:8081/Policy/Trace/external_trace.html
```

Using Trace Information to Improve Policies

To help you understand tracing, this section shows annotated trace output. These traces show the evaluation of specific requests against a particular policy. The sample policy used is not intended as suitable for any particular purpose, other than to illustrate most aspects of policy trace output.

Here are the relevant policy requirements to be expressed:

- DNS lookups are restricted except for a site being hosted.
- There is no access to reverse DNS so that is completely restricted.
- Any requests not addressed to the hosted site either by name or subnet should be rejected.
- FTP POST requests should be rejected.
- Request URLs for the hosted site are to be rewritten and a request header on the way into the site.

The Sample Policy

```
; DNS lookups are restricted except for one site that is being hosted
restrict dns
   .
except
   my_site.com
end

; No access to RDNS
restrict rdns
   all
end

define subnet my_subnet
   10.11.12.0/24
end

<proxy>
   trace.request(yes) trace.rules(all)

proxy>
    ;
   deny url.host.is_numeric=no url.domain=!my_site.com
   deny url.address=!my_subnet

<proxy>
   deny ftp.method=STOR

<proxy>
   url.domain=my_site.com action.test(yes)

define action test
   set(request.x_header.test, “test”)
   rewrite(url, “(.*)\.my_site.com”, “$(1).his_site.com”)
   end
```

Since `trace.request()` is set to `yes`, a policy trace is performed when client requests are evaluated. Since `trace.rules()` is set to `all`, all rule evaluations for misses and matched rules are displayed.

The following is the trace output produced for an HTTP GET request for `http://www.my_site.com/home.html`.

Note: The line numbers shown at the left do not appear in actual trace output. They are added here for annotation purposes.

```
1   start transaction ----------------------------
2     CPL Evaluation Trace:
3               <Proxy>
4       MATCH:     trace.rules(all) trace.request(yes)
5               <Proxy>
6       miss:     url.domain=!//my_site.com/
7       miss:     url.address=!my_subnet
8               <Proxy>
9       n/a  :     ftp.method=STOR
10              <Proxy>
11      MATCH:     url.domain=//my_site.com/ action.foo(yes)
12   connection: client_address=10.10.0.10 proxy_port=36895
13   time: 2003-09-11 19:36:22 UTC
14   GET http://www.my_site.com/home.html
15   DNS lookup was unrestricted
16   rewritten URL(s):
17   cache_url/server_url/log_url=http://www.his_site.com/
18   User-Agent: Mozilla 8.6 (Non-compatible)
19   user: unauthenticated
20   set header= (request)
21    value='test'
22  end transaction ------------------------------
```

Notes:

- Lines 1 and 22 are delimiters indicating where the trace for this transaction starts and ends.
- Line 2 introduces the rule evaluation part of the trace. A rule evaluation part is generated when `trace.rules()` is set to `yes` or `all`.
- Lines 3 to 4 and 10 to 11 show rule matches, and are included when `trace.rules()` is set to either `yes` or `all`.
- Lines 5 to 9 come only with `trace.rules(all)`. That is, `trace.rules(yes)` shows only layers and rules that match. To include rules that do not match, use `trace.rules(all)`.
- Line 9 shows how a rule (containing an FTP specific condition) that is not applicable to this transaction (HTTP) is marked as `n/a`.
- Lines 12 to 21 are generated as a result of `trace.request(yes)`. Using `trace.rules()` without `trace.request(yes)` does *not* result in a trace.
- Line 12 show client related information.
- Line 13 shows the time the transaction was processed.
- Line 14 is a summary of the request line.
- Line 15 indicates that DNS lookup was attempted during evaluation, and was unrestricted. This line only appears if there is a DNS restriction and a DNS lookup was required for evaluation.
- Lines 16 and 17 indicate that the request URL was rewritten, and show the effects.
- Line 19 indicates that the user was not required to authenticate. If authentication had been required, the user identity would be displayed.
- Lines 20 and 21 show the results of the header modification action.

The following is a trace of the same policy, but for a transaction in which the request URL has an IP address instead of a hostname.

```
1   start transaction ------------------------------
2     CPL Evaluation Trace:
3               <Proxy>
4       MATCH:     trace.rules(all) trace.request(yes)
5               <Proxy>
6       miss:      url.host.is_numeric=no
7       miss:      url.address=!my_subnet
8               <Proxy>
9       n/a  :     ftp.method=STOR
10              <Proxy>
11      miss:      url.domain=//my_site.com/
12   connection: client_address=10.10.0.10 proxy_port=36895
13   time: 2003-09-11 19:33:34 UTC
14   GET http://10.11.12.13/home.html
15   DNS lookup was restricted
16   RDNS lookup was restricted
17   User-Agent: Mozilla 8.6 (Non-compatible)
18   user: unauthenticated
19  end transaction --------------------------------
```

This shows many of the same features as the earlier trace, but has the following differences:

- Line 12—The URL requested had a numeric host name.
- Lines 15 and 16—Both DNA and RDNS lookups were restricted for this transaction.
- Line 11—Because RDNS lookups are restricted, the rule missed; no rewrite action was used for the transaction and no rewrite action is reported in the transaction summary (lines 12-18).

Trace output can be used to determine the cause of action conflicts that may be reported in the event log. For example, consider the following policy fragment:

```
<proxy>
trace.request(yes) trace.rules(all)

<proxy> action.set_header_1(yes)
    [Rule] action.set_header_2(yes)
        action.set_header_3(yes)

define action set_header_1
    set(request.x_header.Test, "one")
|end

define action set_header_2
    set(request.x_header.Test, "two")
end

define action set_header_3
    set(request.x_header.Test, "three")
end
```

Because they all set the same header, these actions will conflict. In this example, the conflict is obvious because all the actions are enabled in the same layer. However, conflicts can also arise when actions are enabled by completely independent portions of policy. If an action conflict occurs, one of the actions is dropped and an event log entry is made similar to the following:

```
Policy: Action discarded, 'set_header_1' conflicts with an action already committed
```

The conflict is reflected in the following trace of a request for `//www.my_site.com/home.html`:

```
1  start transaction ------------------------------
2    CPL Evaluation Trace:
3              <Proxy>
4      MATCH:     trace.rules(all) trace.request(yes)
5              <Proxy> action.set_header_1(yes)
6                [Rule] action.set_header_2(yes)
7      MATCH:         action.set_header_1(yes)
8      MATCH:         action.set_header_2(yes)
9      MATCH:         action.set_header_3(yes)
10 connection: client_address=10.10.0.10 proxy_port=36895
11 time: 2003-09-12 15:56:39 UTC
12 GET http://www.my_site.com/home.html
13   User-Agent: Mozilla 8.6 (Non-compatible)
14 user: unauthenticated
15 Discarded Actions:
16   set_header_1
17   set_header_2
18 set header=set_header_3 (request)
19   value='three'
20 end transaction ------------------------------
```

Notes:

- Layer and section guard expressions are indicated in the trace (lines 7 and 8) before any rules subject to the guard (line 9).
- Line 15 indicates that actions were discarded due to conflicts.
- Lines 16 and 17 show the discarded actions.
- Line 18 shows the remaining action, while line 19 shows the effect of the action on the header value.

Appendix C: *Recognized HTTP Headers*

The tables provided in this appendix list all recognized HTTP 1.1 headers and indicate how the Proxy*SG* is able to interact with them. For each header, columns show whether the header appears in request or response forms, and whether the `append( )`, `delete( )`, `rewrite( )`, or `set( )` actions can be used to change the header.

Recognized headers can be used with the `request.header.`*`header_name`*`=` and `response.header.`*`header_name`*`=` conditions. Headers not shown in these tables must be tested with the `request.x_header.`*`header_name`*`=` and `response.x_header.`*`header_name`*`=` conditions. In addition, the following three header fields take address values, so they can be used with the condition `request.header.`*`header_name`*`.address= Client-IP, Host, X-Forwarded-For.`

Table C.1: HTTP Headers Recognized by the Proxy*SG*

Header Field	Request/Response Form	Allowed Actions		
		`rewrite( )` `set( )`	`append( )`	`delete( )`
Accept	Request	X	X	X
Accept-Charset	Request	X	X	X
Accept-Encoding	Request	X	X	X
Accept-Language	Request	X	X	X
Accept-Ranges	Response	X	X	X
Age	Response			
Allow	Request/Response	X	X	X
Authorization	Request			
Cache-Control	Request/Response	X	X	X
Client-IP	Request	X		X
Connection	Request/Response		X	
Content-Encoding	Request/Response		X	
Content-Language	Request/Response		X	
Content-Length	Request/Response			
Content-Location	Request/Response	X		X
Content-MD5	Request/Response			
Content-Range	Request/Response			
Content-Type	Request/Response			
Cookie	Request	X	X	X
Cookie2	Request	X		X
Date	Request/Response			
ETag	Response	X		X
Expect	Request		X	
Expires	Request/Response	X		X
From	Request	X		X
Host	Request			

Table C.1: HTTP Headers Recognized by the ProxySG

If-Match	Request		X	
If-Modified-Since	Request			
If-None-Match	Request		X	
If-Range	Request			
If-Unmodified-Since	Request			
Last-Modified	Request/Response			
Location	Response	X		X
Max-Forwards	Request			
Meter	Request/Response	X		X
Pragma	Request/Response	X		X
Proxy-Authenticate	Response		X	
Proxy-Authorization	Request		X	
Proxy-Connection	Request		X	
Range	Request	X		X
Referer	Request	X		X
Retry-After	Response	X		X
Server	Response	X		X
Set-Cookie	Response	X	X	X
Set-Cookie2	Response	X	X	X
TE	Request		X	
Trailer	Request/Response		X	
Transfer-Encoding	Request/Response		X	
Upgrade	Request/Response		X	
User-Agent	Request	X		X
Vary	Response	X	X	X
Via	Request/Response	X	X	X
Warning	Request/Response	X	X	X
WWW-Authenticate	Response			

The following table lists custom headers that are recognized by the ProxySG.

Table C.2: Custom HTTP Headers Recognized by the ProxySG

Header Field	Request/Response Form	Allowed Actions
Authentication-Info	Response	`append( )`
Front-End-Https	Request/Response	`rewrite( ), set( ), delete( )`
P3P	Request/Response	`rewrite( ), set( ), delete( )`
Refresh	Request/Response	`rewrite( ), set( ), delete( )`
X-BlueCoat-Error	Request/Response	Cannot be modified.
X-BlueCoat-Via	Request/Response	`delete( )`
X-Forwarded-For	Request	`rewrite( ), set( ), delete( )`

Appendix D: *CPL Substitutions*

This appendix lists all substitution variables available in CPL.

To use a variable in CPL, it is expressed as: `$(<field-id>`, such as `$(cs-bodylength)`.
For fields that have both ELFF and CPL tokens, either token can be used. For example, `$(cs-ip)` and `$(proxy.address)` are equivalent.

Note that `$(request.x_header.<x-header-name>)` and `$(response.x_header.<x-header-name>)` are also valid substitutions, but are not included in the tables below, because they have no corresponding ELFF tokens.

The available substitutions are organized in the following categories

- `bytes`
- `connection`
- `instant messaging (im)`
- `req_rsp_line`
- `special_token`
- `status`
- `streaming`
- `time`
- `url`
- `user`
- `ci_request_header`
- `si_response_header`

Category: bytes		
ELFF	**CPL**	**Description**
`cs-bodylength`		Number of bytes in the body (excludes header) sent from client to appliance.
`cs-bytes`		Number of bytes sent from client to appliance.
`cs-headerlength`		Number of bytes in the header sent from client to appliance.
`rs-bodylength`		Number of bytes in the body (excludes header) sent from upstream host to appliance.
`rs-bytes`		Number of bytes sent from upstream host to appliance.
`rs-headerlength`		Number of bytes in the header sent from upstream host to appliance.
`sc-bodylength`		Number of bytes in the body (excludes header) sent from appliance to client.
`sc-bytes`		Number of bytes sent from appliance to client.
`sc-headerlength`		Number of bytes in the header sent from appliance to client.
`sr-bodylength`		Number of bytes in the body (excludes header) sent from appliance to upstream host.

sr-bytes		Number of bytes sent from appliance to upstream host.
sr-headerlength		Number of bytes in the header sent from appliance to upstream host.
Category: connection		
ELFF	**CPL**	**Description**
cs-ip	proxy.address	IP address of the destination of the client's connection.
c-connect-type		The type of connection made by the client to the appliance—'Transparent' or 'Explicit'.
c-dns		Hostname of the client (uses the client's IP address to avoid reverse DNS).
c-ip	client.address	IP address of the client.
r-dns		Hostname from the outbound server URL.
r-ip		IP address from the outbound server URL.
r-port		Port from the outbound server URL.
r-supplier-dns		Hostname of the upstream host (not available for a cache hit).
r-supplier-ip		IP address used to contact the upstream host (not available for a cache hit).
r-supplier-port		Port used to contact the upstream host (not available for a cache hit).
sc-adapter	proxy.card	Adapter number of the client's connection to the appliance.
sc-connection		Unique identifier of the client's connection (for example, SOCKET) .
x-bluecoat-server-connection-socket-errno	server_connection.socket_errno	Error message associated with a failed attempt to connect to an upstream host.
s-computername	proxy.name	Configured name of the appliance.
s-connect-type		Upstream connection type (Direct, SOCKS gateway, etc.).
s-dns		Hostname of the appliance (uses the primary IP address to avoid reverse DNS).
s-ip		IP address of the appliance on which the client established its connection.
s-port	proxy.port	Port of the appliance on which the client established its connection.
s-sitename		Service used to process the transaction.
s-supplier-ip		IP address used to contact the upstream host (not available for a cache hit).
s-supplier-name		Hostname of the upstream host (not available for a cache hit).

x-bluecoat-transaction-id	transaction.id	Unique per-request identifier generated by the appliance (note: this value is not unique across multiple appliances).
x-bluecoat-appliance-name	appliance.name	Configured name of the appliance.
x-bluecoat-appliance-primary-address	appliance.primary_address	Primary IP address of the appliance.
x-bluecoat-proxy-primary-address	proxy.primary_address	Primary IP address of the appliance.
x-client-address		IP address of the client.
x-client-ip		IP address of the client.
Category: im		
ELFF	**CPL**	**Description**
x-im-buddy-id		Instant Messaging buddy ID.
x-im-buddy-name		Instant Messaging buddy display name.
x-im-buddy-state		Instant Messaging buddy state.
x-im-chat-room-id		Instant Messaging identifier of the chat room in use.
x-im-chat-room-members		The list of chat room member IDs.
x-im-chat-room-type		The chat room type, one of 'public' or 'public', and possibly 'invite_only', 'voice' and/or 'conference'.
x-im-client-info		The Instant Messaging client information.
x-im-file-path		Path of the file associated with an instant message.
x-im-file-size		Size of the file associated with an instant message.
x-im-message-opcode	im.message.opcode	The opcode utilized in the instant message.
x-im-message-route		The route of the instant message.
x-im-message-size		Length of the instant message.
x-im-message-text		Text of the instant message.
x-im-message-type		The type of the instant message.
x-im-method		The method associated with the instant message.
x-im-user-id		Instant Messaging user identifier.
x-im-user-name		Display name of the client.
x-im-user-state		Instant Messaging user state.
Category: req_rsp_line		
ELFF	**CPL**	**Description**
cs-method	method	Request method used from client to appliance.
cs-protocol	lient.protocol	Protocol used in the client's request.
cs-request-line		First line of the client's request.

cs-version	request.version	Protocol and version from the client's request; for example, HTTP/1.1.
x-bluecoat-proxy-via-http-version	proxy.via_http_version	Default HTTP protocol version of the appliance without protocol decoration (e.g. 1.1 for HTTP/1.1).
x-bluecoat-redirect-location	redirect.location	Redirect location URL specified by a redirect CPL action.
rs-response-line		First line (a.k.a. status line) of the response from an upstream host to the appliance.
rs-status	response.code	Protocol status code of the response from an upstream host to the appliance.
rs-version	response.version	Protocol and version of the response from an upstream host to the appliance; for example, HTTP/1.1.
sc-status		Protocol status code from appliance to client.
x-bluecoat-ssl-failure-reason	ssl_failure_reason	Upstream SSL negotiation failure reason.
x-cs-http-version	http.request.version	HTTP protocol version of request from the client. Does not include protocol qualifier, for example, 1.1 for HTTP/1.1.
x-cs-socks-ip	socks.destination_address	Destination IP address of a proxied SOCKS request.
x-cs-socks-port	socks.destination_port	Destination port of a proxied SOCKS request.
x-cs-socks-method	socks.method	Method of a proxied SOCKS request.
x-cs-socks-version	socks.version	Version of a proxied SOCKS request.
x-sc-http-status	http.response.code	HTTP response code sent from appliance to client.
x-rs-http-version	http.response.version	HTTP protocol version of response from the upstream host. Does not include protocol qualifier; for example, 1.1 for HTTP/1.1.
x-sc-http-version		HTTP protocol version of response to client. Does not include protocol qualifier; for example, 1.1 for HTTP/1.1.
x-sr-http-version		HTTP protocol version of request to the upstream host. Does not include protocol qualifier; for example, 1.1 for HTTP/1.1.
Category: special_token		
ELFF	**CPL**	**Description**
x-bluecoat-special-amp	amp	The ampersand character.
x-bluecoat-special-apos	apos	The apostrophe character (’).
x-bluecoat-special-cr	cr	Resolves to the carriage return character .
x-bluecoat-special-crlf	crlf	Resolves to a carriage return/line feed sequence.
x-bluecoat-special-empty	empty	Resolves to an empty string.

x-bluecoat-special-esc	esc	Resolves to the escape character (ASCII HEX 1B).
x-bluecoat-special-gt	gt	The greater-than character.
x-bluecoat-special-lf	lf	The line feed character.
x-bluecoat-special-lt	lt	The less-than character.
x-bluecoat-special-quot	quot	The double quote character.
x-bluecoat-special-slash	slash	The forward slash character.
Category: status		
ELFF	**CPL**	**Description**
x-bluecoat-release-id	release.id	The release ID of the ProxySG operating system.
cs-categories		All content categories of the request URL.
cs-categories-external		All content categories of the request URL that are defined by an external service.
cs-categories-policy		All content categories of the request URL that are defined by CPL.
cs-categories-provider		All content categories of the request URL that are defined by the current provider.
cs-categories-qualified		All content categories of the request URL, qualified by the provider of the category.
cs-category		Single content category of the request URL (sc-filter-category).
r-hierarchy		How and where the object was retrieved in the cache hierarchy.
sc-filter-category	category	Content filtering category of the request URL.
sc-filter-result		Content filtering result: Denied, Proxied or Observed.
s-action		What type of action the Appliance took to process this request.
s-cpu-util		Average load on the proxy's processor (0%-100%).
s-hierarchy		How and where the object was retrieved in the cache hierarchy.
s-icap-info		ICAP response information.
s-icap-status		ICAP response status.
x-bluecoat-surfcontrol-category-id		The SurfControl specific content category ID.
x-bluecoat-surfcontrol-is-denied		1 if the transaction was denied, else 0.
x-bluecoat-surfcontrol-is-proxied		0 if transaction is explicitly proxied, 1 if transaction is transparently proxied.

x-bluecoat-surfcontrol-reporter-id		Specialized value for SurfControl reporter.
x-bluecoat-websense-category-id		The Websense specific content category ID.
x-bluecoat-websense-keyword		The Websense specific keyword.
x-bluecoat-websense-reporter-id		The Websense specific reporter category ID.
x-bluecoat-websense-status		The Websense specific numeric status.
x-bluecoat-websense-user		The Websense form of the username.
x-exception-company-name	exception.company_name	The company name configured under exceptions.
x-exception-contact	exception.contact	Describes who to contact when certain classes of exceptions occur, configured under exceptions (empty if the transaction has not been terminated).
x-exception-details	exception.details	The configurable details of a selected policy-aware response page (empty if the transaction has not been terminated).
x-exception-header	exception.header	The header to be associated with an exception response (empty if the transaction has not been terminated).
x-exception-help	exception.help	Help text that accompanies the exception resolved (empty if the transaction has not been terminated).
x-exception-id	exception.id	Identifier of the exception resolved (empty if the transaction has not been terminated).
x-exception-last-error	exception.last_error	The last error recorded for the current transaction. This can provide insight when unexpected problems are occurring (empty if the transaction has not been terminated).
x-exception-reason	exception.reason	Indicates the reason why a particular request was terminated (empty if the transaction has not been terminated).
x-exception-sourcefile	xception.sourcefile	Source filename from which the exception was generated (empty if the transaction has not been terminated).
x-exception-sourceline	exception.sourceline	Source file line number from which the exception was generated (empty if the transaction has not been terminated).
x-exception-summary	xception.summary	Summary of the exception resolved (empty if the transaction has not been terminated).
x-patience-javascript	patience_javascript	Javascript required to allow patience responses.
x-patience-progress	patience_progress	The progress of the patience request.
x-patience-time	patience_time	The elapsed time of the patience request.

x-patience-url	patience_url	The url to be requested for more patience information.
x-virus-id		Identifier of a virus if one was detected.
Category: streaming		
ELFF	**CPL**	**Description**
x-cs-streaming-client	streaming.client	Type of streaming client in use (windows_media, real_media, or quicktime).
x-rs-streaming-content	streaming.content	Type of streaming content served; for example, windows_media, quicktime).
x-streaming-bitrate	bitrate	The reported client-side bitrate for the stream.
Category: time		
ELFF	**CPL**	**Description**
connect-time		Total MS required to connect to the origin server.
date	date.utc	GMT Date in YYYY-MM-DD format.
dnslookup-time		Total ms cache required to perform the DNS lookup.
duration		Time taken (in seconds) to process the request.
gmttime		GMT date and time of the user request in format: [DD/MM/YYYY:hh:mm:ss GMT].
x-bluecoat-day-utc	day.utc	GMT/UTC day (as a number) formatted to take up two spaces; for example, 07 for the 7th of the month.
x-bluecoat-hour-utc	hour.utc	GMT/UTC hour formatted to always take up two spaces; for example, 01 for 1AM.
x-bluecoat-minute-utc	minute.utc	GMT/UTC minute formatted to always take up two spaces; for example, 01 for 1 minute past.
x-bluecoat-month-utc	month.utc	GMT/UTC month (as a number) formatted to take up two spaces; for example, 01 for January.
x-bluecoat-monthname-utc	monthname.utc	GMT/UTC month in the short-form string representation; for example, Jan for January.
x-bluecoat-second-utc	second.utc	GMT/UTC second formatted to always take up two spaces; for example, 01 for 1 second past.
x-bluecoat-weekday-utc	weekday.utc	GMT/UTC weekday in the short-form string representation; for example, Mon for Monday.
x-bluecoat-year-utc	year.utc	GMT/UTC year formatted to always take up four spaces.
localtime		Local date and time of the user request in format: [DD/MMM/YYYY:hh:mm:ss +nnnn].

x-bluecoat-day	day	Localtime day (as a number) formatted to take up two spaces; for example, 07 for the 7th of the month.
x-bluecoat-hour	hour	Localtime hour formatted to always take up two spaces; for example, 01 for 1AM.
x-bluecoat-minute	minute	Localtime minute formatted to always take up two spaces; for example, 01 for 1 minute past.
x-bluecoat-month	month	Localtime month (as a number) formatted to take up two spaces; for example, 01 for January.
x-bluecoat-monthname	monthname	Localtime month in the short-form string representation; for example, Jan for January.
x-bluecoat-second	second	Localtime second formatted to always take up two spaces; for example, 01 for 1 second past.
x-bluecoat-weekday	weekday	Localtime weekday in the short-form string representation; for example, Mon for Monday.
x-bluecoat-year	year	Localtime year formatted to always take up four spaces.
time	time.utc	GMT time in HH:MM:SS format.
timestamp		Unix-type timestamp.
time-taken		Time taken (in milliseconds) to process the request.
x-bluecoat-end-time-wft		End local time of the transaction represented as a windows file time.
x-bluecoat-start-time-wft		Start local time of the transaction represented as a windows file time.
x-cookie-date	cookie_date	Current date in Cookie time format.
x-http-date	http_date	Current date in HTTP time format.
x-timestamp-unix		Seconds since UNIX epoch (Jan 1, 1970) (local time) .
x-timestamp-unix-utc		Seconds since UNIX epoch (Jan 1, 1970) (GMT/UTC).
Category: url		
ELFF	**CPL**	**Description**
cs-host		Hostname from the client's request URL. If URL rewrite policies are used, this field's value is derived from the 'log' URL. See x-s-log-uri-host.
cs-uri	log_url	The 'log' URL.
cs-uri-address	log_url.address	IP address from the 'log' URL. DNS is used if URL uses a hostname.
cs-uri-extension	log_url.extension	Document extension from the 'log' URL.
cs-uri-host	log_url.host	Hostname from the 'log' URL.

cs-uri-hostname	log_url.hostname	Hostname from the 'log' URL. RDNS is used if the URL uses an IP address.
cs-uri-path	log_url.path	Path from the 'log' URL. Does not include query.
cs-uri-pathquery	log_url.pathquery	Path and query from the 'log' URL.
cs-uri-port	log_url.port	Port from the 'log' URL.
cs-uri-query	log_url.query	Query from the 'log' URL.
cs-uri-scheme	log_url.scheme	Scheme from the 'log' URL.
cs-uri-stem		Stem from the 'log' URL. The stem includes everything up to the end of path, but does not include the query.
c-uri	url	The original URL requested.
c-uri-address	url.address	IP address from the original URL requested. DNS is used if the URL is expressed as a hostname.
c-uri-cookie-domain	url.cookie_domain	The cookie domain of the original URL requested
c-uri-extension	url.extension	Document extension from the original URL requested
c-uri-host	url.host	Hostname from the original URL requested
c-uri-hostname	url.hostname	Hostname from the original URL requested. RDNS is used if the URL is expressed as an IP address
c-uri-path	url.path	Path of the original URL requested without query.
c-uri-pathquery	url.pathquery	Path and query of the original URL requested
c-uri-port	url.port	Port from the original URL requested
c-uri-query	url.query	Query from the original URL requested
c-uri-scheme	url.scheme	Scheme of the original URL requested
c-uri-stem		Stem of the original URL requested
sr-uri	server_url	URL of the upstream request
sr-uri-address	server_url.address	IP address from the URL used in the upstream request. DNS is used if the URL is expressed as a hostname.
sr-uri-extension	server_url.extension	Document extension from the URL used in the upstream request
sr-uri-host	server_url.host	Hostname from the URL used in the upstream request
sr-uri-hostname	server_url.hostname	Hostname from the URL used in the upstream request. RDNS is used if the URL is expressed as an IP address.
sr-uri-path	server_url.path	Path from the upstream request URL
sr-uri-pathquery	server_url.pathquery	Path and query from the upstream request URL
sr-uri-port	server_url.port	Port from the URL used in the upstream request.

sr-uri-query	server_url.query	Query from the upstream request URL.
sr-uri-scheme	server_url.scheme	Scheme from the URL used in the upstream request.
sr-uri-stem		Path from the upstream request URL
s-uri	cache_url	The URL used for cache access.
s-uri-address	cache_url.address	IP address from the URL used for cache access. DNS is used if the URL is expressed as a hostname.
s-uri-extension	cache_url.extension	Document extension from the URL used for cache access.
s-uri-host	cache_url.host	Hostname from the URL used for cache access.
s-uri-hostname	cache_url.hostname	Hostname from the URL used for cache access. RDNS is used if the URL uses an IP address.
s-uri-path	cache_url.path	Path of the URL used for cache access
s-uri-pathquery	cache_url.pathquery	Path and query of the URL used for cache access.
s-uri-port	cache_url.port	Port from the URL used for cache access.
s-uri-query	cache_url.query	Query string of the URL used for cache access.
s-uri-scheme	cache_url.scheme	Scheme from the URL used for cache access.
s-uri-stem		Stem of the URL used for cache access.
x-cs(Referer)-uri	request.header.Referer.url	The URL from the Referer header.
x-cs(Referer)-uri-address	request.header.Referer.url.address	IP address from the 'Referer' URL. DNS is used if URL uses a hostname.
x-cs(Referer)-uri-extension	request.header.Referer.url.extension	Document extension from the 'Referer' URL.
x-cs(Referer)-uri-host	request.header.Referer.url.host	Hostname from the 'Referer' URL.
x-cs(Referer)-uri-hostname	request.header.Referer.url.hostname	Hostname from the 'Referer' URL. RDNS is used if the URL uses an IP address.
x-cs(Referer)-uri-path	request.header.Referer.url.path	Path from the 'Referer' URL. Does not include query.
x-cs(Referer)-uri-pathquery	request.header.Referer.url.pathquery	Path and query from the 'Referer' URL.
x-cs(Referer)-uri-port	request.header.Referer.url.port	Port from the 'Referer' URL.
x-cs(Referer)-uri-query	request.header.Referer.url.query	Query from the 'Referer' URL.
x-cs(Referer)-uri-scheme	request.header.Referer.url.scheme	Scheme from the 'Referer' URL.
x-cs(Referer)-uri-stem		Stem from the 'Referer' URL. The stem includes everything up to the end of path, but does not include the query.

Category: user		
ELFF	**CPL**	**Description**
cs-auth-group	group	One group that an authenticated client is a member of. The group selected is determined by either a group.log_order definition in policy or the order groups are referenced in policy
cs-auth-groups	groups	Groups that an authenticated client is a member of.
cs-auth-type		Client-side: authentication type (such as BASIC, NTLM, LDAP)
cs-realm	realm	Authentication realm that the user was challenged in.
cs-userdn	user	Full username of a client authenticated to the proxy (fully distinguished).
cs-username	user.name	Relative username of a client authenticated to the proxy; for example, not fully distinguished.
sc-auth-status		Client-side: Authorization status.
x-cache-user		Relative username of a client authenticated to the proxy; for example, not fully distinguished (same as cs-username).
x-cs-username-or-ip		Used to identify the user using either their authenticated proxy username or, if that is unavailable, their IP address.
x-radius-splash-session-id		Session ID made available through RADIUS when configured for session management
x-radius-splash-username		Username made available through RADIUS when configured for session management
x-user-x509-issuer	user.x509.issuer	If the user was authenticated through an X.509 certificate, this is the issuer of the certificate as an RFC2253 DN.
x-user-x509-serial-number	user.x509.serialNumber	If the user was authenticated through an X.509 certificate, this is the serial number from the certificate as a hexadecimal number.
x-user-x509-subject	user.x509.subject	If the user was authenticated through an X.509 certificate, this is the subject of the certificate as an RFC2253 DN.
Category: ci_request_header		
ELFF	**CPL**	**Description**
cs(Accept)	request.header.Accept	Request header: Accept
cs(Accept-Charset)	request.header.Accept-Charset	Request header: Accept-Charset
cs(Accept-Encoding)	request.header.Accept-Encoding	Request header: Accept-Encoding

cs(Accept-Language)	request.header.Accept-Language	Request header: Accept-Language
cs(Accept-Ranges)	request.header.Accept-Ranges	Request header: Accept-Ranges
cs(Age)	request.header.Age	Request header: Age
cs(Allow)	request.header.Allow	Request header: Allow
cs(Authentication-Info)	request.header.Authentication-Info	Request header: Authentication-Info
cs(Authorization)	request.header.Authorization	Request header: Authorization
cs(Cache-Control)	request.header.Cache-Control	Request header: Cache-Control
cs(Client-IP)	request.header.Client-IP	Request header: Client-IP
cs(Connection)	request.header.Connection	Request header: Connection
cs(Content-Encoding)	request.header.Content-Encoding	Request header: Content-Encoding
cs(Content-Language)	request.header.Content-Language	Request header: Content-Language
cs(Content-Length)	request.header.Content-Length	Request header: Content-Length
cs(Content-Location)	request.header.Content-Location	Request header: Content-Location
cs(Content-MD5)	request.header.Content-MD5	Request header: Content-MD5
cs(Content-Range)	request.header.Content-Range	Request header: Content-Range
cs(Content-Type)	request.header.Content-LType	Request header: Content-Type
cs(Cookie)	request.header.Cookie	Request header: Cookie
cs(Cookie2)	request.header.Cookie2	Request header: Cookie2
cs(Date)	request.header.Date	Request header: Date
cs(Etag)	request.header.Etag	Request header: Etag
cs(Expect)	request.header.Expect	Request header: Expect
cs(Expires)	request.header.Expires	Request header: Expires
cs(From)	request.header.From	Request header: From
cs(Front-End-HTTPS)	request.header.Front-End-HTTPS	Request header: Front-End-HTTPS
cs(Host)	request.header.Host	Request header: Host
cs(If-Match)	request.header.If-Match	Request header: If-Match
cs(If-Modified-Since)	request.header.If-Modified-Since	Request header: If-Modified-Since
cs(If-None-Match)	request.header.If-None-Match	Request header: If-None-Match
cs(If-Range)	request.header.If-Range	Request header: If-Range

cs(If-Unmodified-Since)	request.header.If-Unmodified-Since	Request header: If-Unmodified-Since
cs(Last-Modified)	request.header.Last-Modified	Request header: Last-Modified
cs(Location)	request.header.Location	Request header: Location
cs(Max-Forwards)	request.header.Max-Forwards	Request header: Max-Forwards
cs(Meter)	request.header.Meter	Request header: Meter
cs(P3P)	request.header.P3P	Request header: P3P
cs(Pragma)	request.header.Pragma	Request header: Pragma
cs(Proxy-Authenticate)	request.header.Proxy-Authenticate	Request header: Proxy-Authenticate
cs(Proxy-Authorization)	request.header.Proxy-Authorization	Request header: Proxy-Authorization
cs(Proxy-Connection)	request.header.Proxy-Connection	Request header: Proxy-Connection
cs(Range)	request.header.Range	Request header: Range
cs(Referer)	request.header.Referer	Request header: Referer
cs(Refresh)	request.header.Refresh	Request header: Refresh
cs(Retry-After)	request.header.Retry-After	Request header: Retry-After
cs(Server)	request.header.Server	Request header: Server
cs(Set-Cookie)	request.header.Set-Cookie	Request header: Set-Cookie
cs(Set-Cookie2)	request.header.Set-Cookie2	Request header: Set-Cookie2
cs(TE)	request.header.TE	Request header: TE
cs(Trailer)	request.header.Trailer	Request header: Trailer
cs(Transfer-Encoding)	request.header.Transfer-Encoding	Request header: Transfer-Encoding
cs(Upgrade)	request.header.Upgrade	Request header: Upgrade
cs(User-Agent)	request.header.User-Agent	Request header: User-Agent
cs(Vary)	request.header.Vary	Request header: Vary
cs(Via)	request.header.Via	Request header: Via
cs(WWW-Authenticate)	request.header.WWW-Authenticate	Request header: WWW-Authenticate
cs(Warning)	request.header.Warning	Request header: Warning
cs(X-BlueCoat-Error)	request.header.X-BlueCoat-Error	Request header: X-BlueCoat-Error
cs(X-BlueCoat-MC-Client-Ip)	request.header.X-BlueCoat-MC-Client-Ip	Request header: X-BlueCoat-MC-Client-Ip
cs(X-BlueCoat-Via)	request.header.X-BlueCoat-Via	Request header: X-BlueCoat-Via

cs(X-Forwarded-For)	request.header. X-Forwarded-For	Request header: X-Forwarded-For
Category: si_response_header		
ELFF	**CPL**	**Description**
rs(Accept)	response.header.Accept	Response header: Accept
rs(Accept-Charset)	response.header. Accept-Charset	Response header: Accept-Charset
rs(Accept-Encoding)	response.header. Accept-Encoding	Response header: Accept-Encoding
rs(Accept-Language)	response.header. Accept-Language	Response header: Accept-Language
rs(Accept-Ranges)	response.header. Accept-Ranges	Response header: Accept-Ranges
rs(Age)	response.header.Age	Response header: Age
rs(Allow)	response.header.Allow	Response header: Allow
rs(Authentication- Info)	response.header. Authentication-Info	Response header: Authentication-Info
rs(Authorization)	response.header. Authorization	Response header: Authorization
rs(Cache-Control)	response.header. Cache-Control	Response header: Cache-Control
rs(Client-IP)	response.header. Client-IP	Response header: Client-IP
rs(Connection)	response.header. Connection	Response header: Connection
rs(Content-Encoding)	response.header. Content-Encoding	Response header: Content-Encoding
rs(Content-Language)	response.header. Content-Language	Response header: Content-Language
rs(Content-Length)	response.header. Content-Length	Response header: Content-Length
rs(Content-Location)	response.header. Content-Location	Response header: Content-Location
rs(Content-MD5)	response.header. Content-MD5	Response header: Content-MD5
rs(Content-Range	response.header. Content-Range	Response header: Content-Range
rs(Content-Type)	response.header. Content-Type	Response header: Content-Type
rs(Cookie)	response.header.Cookie	Response header: Cookie
rs(Cookie2)	response.header.Cookie2	Response header: Cookie2
rs(Date)	response.header.Date	Response header: Date
rs(Etag)	response.header.Etag	Response header: Etag
rs(Expect)	response.header.Expect	Response header: Expect
rs(Expires)	response.header.Expires	Response header: Expires

rs(From)	response.header.From	Response header: From
rs(Front-End-HTTPS)	response.header. Front-End-HTTPS	Response header: Front-End-HTTPS
rs(Host)	response.header.Host	Response header: Host
rs(If-Match)	response.header. If-Match	Response header: If-Match
rs(If-Modified-Since)	response.header. If-Modified-Since	Response header: If-Modified-Since
rs(If-None-Match)	response.header. If-None-Match	Response header: If-None-Match
rs(If-Range)	response.header. If-Range	Response header: If-Range
rs(If-Unmodified- Since)	response.header. If-Unmodified-Since	Response header: If-Unmodified-Since
rs(Last-Modified)	response.header. Last-Modified	Response header: Last-Modified
rs(Location)	response.header. Location	Response header: Location
rs(Max-Forwards)	response.header. Max-Forwards	Response header: Max-Forwards
rs(Meter)	response.header.Meter	Response header: Meter
rs(P3P)	response.header.P3P	Response header: P3P
rs(Pragma)	response.header.Pragma	Response header: Pragma
rs(Proxy-Authenticate)	response.header. Proxy-Authenticate	Response header: Proxy-Authenticate
rs(Proxy- Authorization)	response.header. Proxy-Authorization	Response header: Proxy-Authorization
rs(Proxy-Connection)	response.header. Proxy-Connection	Response header: Proxy-Connection
rs(Range)	response.header.Range	Response header: Range
rs(Referer)	response.header.Referer	Response header: Referer
rs(Refresh)	response.header.Refresh	Response header: Refresh
rs(Retry-After)	response.header. Retry-After	Response header: Retry-After
rs(Server)	response.header.Server	Response header: Server
rs(Set-Cookie)	response.header. Set-Cookie	Response header: Set-Cookie
rs(Set-Cookie2)	response.header. Set-Cookie2	Response header: Set-Cookie2
rs(TE)	response.header.TE	Response header: TE
rs(Trailer)	response.header.Trailer	Response header: Trailer
rs(Transfer-Encoding)	response.header. Transfer-Encoding	Response header: Transfer-Encoding
rs(Upgrade)	response.header.Upgrade	Response header: Upgrade
rs(User-Agent)	response.header. User-Agent	Response header: User-Agent

rs(Vary)	response.header.Vary	Response header: Vary
rs(Via)	response.header.Via	Response header: Via
rs(WWW-Authenticate)	response.header. WWW-Authenticate	Response header: WWW-Authenticate
rs(Warning)	response.header.Warning	Response header: Warning
rs(X-BlueCoat-Error)	response.header. X-BlueCoat-Error	Response header: X-BlueCoat-Error
rs(X-BlueCoat-MC-Client-Ip)	response.header. X-BlueCoat-MC-Client-Ip	Response header: X-BlueCoat-MC-Client-Ip
rs(X-BlueCoat-Via)	response.header. X-BlueCoat-Via	Response header: X-BlueCoat-Via
rs(X-Forwarded-For)	response.header. X-Forwarded-For	Response header: X-Forwarded-For

Appendix E: *Filter File Syntax*

This appendix provides a summary of the syntax and evaluation order used in CacheOS version 4.*x* filter files. While it is recommended that you convert any filter file to take advantage of the policy features of Proxy*SG*, it is possible to use a CacheOS 4.*x* filter file in the place of a policy file, and have it work with a few differences. However, using a CacheOS 4.*x* filter file causes deprecation warnings to be emitted by the CPL compiler. For more information about modifications needed to use a filter file with Proxy*SG*, see the "Upgrading and Downgrading" section of Chapter 1.

Filter File Overview

The Proxy*SG* can filter requests made by clients using a filter list. When a filter list is loaded, all requested URLs are compared to the list and processed based on the results.

A filter list can be used to assign the following actions for a URL:

- access direct
- bypass LDAP authentication
- cache advertising objects
- case-insensitive matching
- content-filter override
- deny service
- do not cache
- do not refresh
- time to live (TTL)
- version control
- URL rewriting
- Active Content Management

Important: The Proxy*SG* does not evaluate items in a filter file by the order in which they appear; instead, prefix filters are evaluated first, then domain suffix filters, and lastly, regular expression filters. For more details about evaluation, see "Evaluation Order" on page 306.

Filter File Structure

A CacheOS 4.*x* filter file consists of two parts, both of which are optional. The two parts are divided by a `define_actions` line. The first part, which can be considered the *filter part*, consists of filters and access-control list (ACL) definitions. The second part, or *action part*, contains action and transformer definitions. All filters must be written above the `define_actions` line. All action and transformer definitions must be written below the `define_actions` line.

By contrast, CPL action and transformer definitions may appear anywhere in the policy file.

Filter-Part Components

The filter part of a filter file can contain the following:

- Filters that are not part of a section
- Sections
- ALL statements
- `default_filter_properties` statements
- Access-control list (ACL) definitions

Filters that are not part of a section must occur before the first section. The `default_filter_properties` statements must be written after the last filter or section. The ALL statements and ACL definitions can be written anywhere before the `define_actions` line. All of these components are optional.

Filters

In CPL, the concept of a filter has been replaced by the concept of a *rule*.

A filter is a line that includes, at a minimum, a URL pattern. The filter is considered to be a match if the requested URL matches the URL pattern. It can also include a tag specifying whether the match will be case-sensitive, an `acl` condition expression for specifying a defined access-control list, and a property setting. Multiple `acl` conditions and property settings can be listed. A filter line has the following general syntax:

```
url_pattern [case_insensitive = { yes | no }] [acl=expression] [property=value]
...
```

`url_pattern`

where *`url_pattern`* is either a prefix-style pattern (like the *`prefix_pattern`* used in the `url=` condition) or a regular-expression pattern (as is used in the `url_regex=` condition, see "Sections" on page 303). For more information on URLs, see "url=" on page 137.

`case_insensitive= {yes|no}`

where `case_insensitive` is an optional property that can specify whether URLs matches are case-sensitive. By default, matching is case-sensitive. For more information, see "Properties" on page 301.

`acl=`*`expression`*

where `acl=` can include an IP address or subnet, or the label of a `define acl` definition block. For more information, see "Conditions" on page 301.

`property=value`

where `property=` is an optional property setting. For a list of properties available in filter files, see "Properties" on page 301.

The following are differences with CPL:

- Property settings have the form *`property=value`* in filter files, instead of the CPL form *`property(value)`*.

- The only condition available in filter lines is the `acl=` condition, which is a synonym for the CPL condition `client.address=`.
- The only way to specify case-sensitivity is with `case_insensitive={yes|no}`.

The following are requirements for filter lines:

- A line break is considered to be a new filter line.
- Each line lists a unique URL.
- Comment lines begin with a semicolon (`;`).
- The maximum length of a line is 4096 bytes.

Important: If you include a period at the beginning of the domain name in a filter, it might not produce the expected match, for example, `.company.com` will not match `company.com`. This also holds true for filters that specify only the ending part of the domain name; for example, `org` works as expected, but `.org` does not work as you might expect. If you are using a regular-expression pattern for the filter, a period can be matched by using "\." For more information about using regular expressions, refer to Appendix E: "Using Regular Expressions," in the Proxy*SG Configuration and Management Guide*.

Conditions

In CacheOS 4.x filter files, the only condition is the `acl=` condition. This condition can be used in a filter line to test the IP address of the client. The expression can include an IP address or subnet, or the label of a `define acl` definition block. (In CPL, this condition is deprecated; use the synonym `client.address=` condition along with the `define subnet` definition block.)

`acl={`*`ip_address`*`|`*`acl_label`*`}`

where :

- *`ip_address`* —The client IP address or subnet; for example, `10.1.198.0`.
- *`subnet_label`*—Label of a define ACL definition block that binds a number of IP addresses or subnets.

Properties

Properties in filter files take the following general form: `property=value`. The following table lists the property settings that are available.

Table F.1: Properties available in CacheOS 4.*x* filter files

Property	Value	Description
`always_verify`	`yes | no`	When set to `yes`, acts as the equivalent of `always-verify-source` configurable through caching settings, but on a per-URL basis instead of globally. A verification is performed with the origin server for every request matching the filter. If there are multiple simultaneous accesses of an object, the requests are reduced to a single request to the origin server.
`advertisement`	`yes | no`	When set to `yes`, cache objects at this URL, and request the objects in the background to maintain the hit count.

Table F.1: Properties available in CacheOS 4.*x* filter files

`cache`	`yes | no`	When set to `no`, do not cache the object. When set to `yes`, cache certain objects that would not normally be cached. For more information, see "force_cache( )" on page 180.
`case_insensitive`	`yes | no`	When set to `yes`, match URLs without regard to case-sensitivity. By default, all URLs are matched in a case-sensitive manner. This filter should be set to match URLs served by operating systems such as Windows, which is case-insensitive. If case-insensitivity is to be used with a regular expression, you must use `(?i)` to start the expression to be evaluated. Note: In CPL, `url=` conditions have an optional `.case_sensitive` modifier.
`direct`	`yes | no`	When set to `yes`, do not forward requests to a parent proxy or SOCKS server. This property only applies when the device is configured to forward requests.
`label`	`label_name`	Invokes a labeled definition. Acceptable characters are: [a-zA-Z][a-zA-Z0-9]* Note: In CPL, use the `action( )` property. `Label( )` is deprecated.
`prefetch`	`yes | no`	When set to `yes`, forces pipelining for an object. Set to `no` to prevent the object from being pipelined. The default value is `yes`. Note: In CPL, use the synonym `pipeline( )` property. `Prefetch( )` is deprecated.
`proxy_authentication`	`yes | no`	When set to `no`, bypasses authentication for the URLs specified. Note: In CPL, use the `authenticate( )` property. `Proxy_authentication( )` is deprecated.
`refresh`	`yes | no`	When set to `no`, do not refresh the object if it is cached.
`service`	`yes | no`	When set to `no`, deny service to the URL. Note: In CPL, use `allow` and `deny`. `Service( )` is deprecated.
`ttl`	`seconds`	Sets the expiration time of a URL or object. Notes: • The `advertisement` property overrides the TTL. • The HTTP command-line option "`Force explicit expirations: Never serve after`" must be enabled. If disabled, the CacheOS probabilistic refresh overrides the TTL value.

ALL Statements

An ALL statement is a line beginning with the keyword `ALL`, followed by zero or more conditions and property settings. There are two conditions available in an ALL statement: `acl=` and `protocol=`. The ALL statement acts as a match of first resort, before any filters are matched. An ALL statement has the following general syntax:

```
ALL [acl=expression] [protocol=identifier] [property=value] ...
```

where

- `acl=`*`expression`*—An optional `acl=` condition expression. For more information, see "Conditions" on page 301.

- `protocol=`*`value`*—An optional `protocol=` condition expression. Available values are `http`, `https`, `ftp`, `mms`, `rtsp`, `tcp`, `aol-im`, `msn-im`, or `yahoo-im`. For details, see "url=" on page 137.
- *`property=value`*—An optional property setting. For a list of properties available in filter files, see Table 1, "Properties available in CacheOS 4.x filter files," on page 301.

Access-Control List (ACL) Definitions

The only definition appearing in the filter part of a filter file is the `define acl` definition block, which defines access-control lists. It does this by binding a user-defined label to a set of IP addresses or IP subnet patterns. This label can then be used in an `acl=` expression on a filter line.

This definition block has the same syntax and semantics as a CPL `define subnet` definition block, except that the keyword `subnet` is replaced by the keyword `acl`. The IP addresses or subnets are considered to have a Boolean OR relationship, no matter whether they are all on one line or separate lines. The syntax for the `define acl` definition block is as follows:

```
define acl label
{ip_address|subnet} {ip_address|subnet}...
...
end acl label
```

`where:`

- *`label`*—A user-defined identifier for this subnet definition.
- *`ip_address`*—IP address; for example, `10.1.198.0.`
- *`subnet`*—Subnet specification; for example, `10.25.198.0/16`.

Sections

Filter files support three kinds of sections:

- Prefix sections, for prefix-pattern filters (CPL equivalent: [url]).
- Domain-suffix sections, for domain-suffix filters (CPL equivalent: [url.domain]).
- Regular-expression sections, for regular-expression filters (CPL equivalent: [url.regex]).

A section within a filter file is similar to the equivalent section that appears in a standard CPL policy file; however, filter-file sections do not support guard expressions, and they cannot include `[Rule]` sections.

The appearance of a section header within a filter file indicates that all subsequent filter entries are to be interpreted as specified within the section header. In addition, sections may contain ALL statements and `define acl` definition blocks, but these do not affect the semantics of the section or the way in which the ALL statement and definitions are evaluated.

Note that in the absence of filter section headers, filters are considered to be prefix filters unless they contain one or more regular expression metacharacters. If a filter entry does contain regular expression metacharacters, it is considered to be a regular expression. If section headers are used, the Proxy*SG* automatically checks to ensure that regular expression filter entries only appear within the `[Regular-Expression]` filter section.

Prefix Sections

While prefix-pattern filters are commonly used outside of any section, the Prefix section is provided to help differentiate these type of filters when domain-suffix and regular-expression filters are also used. The filters in a prefix section follow the pattern used in a CPL `url=` condition. For more information, see "url=" on page 137.

Prefix section headers have the following syntax. They are not case-sensitive.

```
[Prefix]
```

Note: In CPL, use `[url]` sections. `[Prefix]` sections are deprecated.

Domain-Suffix Sections

If the filter file includes domain-suffix filters, then those filter lines must be placed within a domain-suffix section. Domain-suffix filters can be used in place of certain regular expression filters and provide better performance than the equivalent regular-expression filters. Domain-suffix filters are intended to replace regular expression filters of the form: **http://.*\.?***domain***/** and match all objects from the domain and its sub-domains. Proxy*SG* supports a filter list containing many domain-suffix filters with minimal system overhead.

Domain-suffix section headers have the following syntax. They are not case-sensitive.

```
[Domain-Suffix]
```

Note: In CPL, use `[url.domain]` sections. `[Domain-Suffix]` sections are deprecated.

Regular-Expression Sections

Regular-expression filters are powerful but they are difficult to write correctly and have a performance penalty. The domain-suffix section is provided to improve performance when processing domain-suffix-style regular-expressions. The filters in a regular-expression section follow the pattern used in a CPL `url.regex=` condition. For more information, see "url=" on page 137.

Regular-expression section headers have the following syntax. They are not case-sensitive.

```
[Regular-Expression]
```

Note: In CPL, use `[Rule]` sections and `url.regex=` conditions. `[url.regex]` sections are supported in CPL and are equivalent to filter file `[Regular-Expression]` sections, but provide no performance advantage. `[Regular-Expression]` sections are deprecated.

Section Example

The following example shows a filter list containing all three types of sections. Filter lists that include domain-suffix filters must follow a structure that explicitly identifies the filter types.

```
[Prefix]
   http://www.confidential.com/ deny
[Domain-Suffix]
   http://company.com/ deny
[Regular-Expression]
   http://.*xyz.com/ deny
```

The above three filter lines all result in denial of service to a group of distinct URLs:

- The prefix filter `http://www.confidential.com/` denies service to all URLs exactly matching the domain `www.confidential.com` and any path relative to the aforementioned domain, including the null path.

- The domain-suffix filter `http://company.com/` denies service to all URLs where `company.com` is a proper super-domain and any path relative to the matched domain, including the null path. For example, service is denied to the URL `http://www.intranet.company.com/`, but not `http://mycompany.com/` since `mycompany.com` is not a proper subdomain of `company.com`.
- The regular expression filter line `http://.*xyz.com/` will deny service to any URL containing a domain ending in the string `xyz.com`. Regular expression filters should only be used when prefix or domain suffix filters are insufficient since processing of regular expression filters requires more system resources.

default_filter_properties Statement

A `default_filter_properties` statement consists of the keyword `default_filter_properties` followed by one or more property settings. This statement acts as the match of last resort, and it must follow any filters or sections. This statement has the following syntax:

```
default_filter_properties property=value ...
```

where *`property=value`* is a property setting. For a list of properties available in filter files, see Table 1, “Properties available in CacheOS 4.x filter files,” on page 301.

Action-Part Components

The action part of a filter file contains all action and transformer definitions used in the filter file. The actions available are limited to `replace( )`, which has been deprecated in CPL in favour of a `rewrite( )` action targeting the URL.

Active content transformers are available but use the following syntax:

```
transform active_content transformer_id
{
  tag_replace HTML_tag_name << text_end_delimiter
  [replacement_text]
  text_end_delimiter
  [tag_replace ...]
  ...
}
```

Where the body of the definition has the same form as the CPL `define active_content` definition block.

URL rewrite transformers are available but use the following syntax:

```
transform url_rewrite transformer_id
{
  [caseless]
  subst_embedded “external_URL_prefix” “internal_URL_prefix”
  ...
}
```

Where the body of the definition has the same form as the CPL `define url_rewrite` definition block.

Evaluation Order

CacheOS 4.*x* filter files have a different order of evaluation than CPL files.

A compiled filter file behaves as if it had a single `[Prefix]` section, a single `[Domain-Suffix]` section, and a single `[Regular-Expression]` section. The filter file is rewritten during file compilation, as follows:

- Any naked filter line that contains regular-expression metacharacters is moved into a *virtual* `[Regular-Expression]` section.
- Any remaining naked filter lines are moved into a virtual `[Prefix]` section.
- All explicit `[Prefix]` sections are appended to the virtual `[Prefix]` section, in the order they are written.
- All explicit `[Domain-Suffix]` sections are appended to the virtual `[Domain-Suffix]` section, in the order they are written.
- All explicit `[Regular-Expression]` sections are appended to the virtual `[Regular-Expression]` section, in the order they are written.

After all of this rewriting is performed, the filter file has the following order of evaluation:

1. The ALL statements.
2. The virtual `[Prefix]` section. Within this section, it is the longest match that *wins*, not the first match.
3. The virtual `[Domain-Suffix]` section. Within this section, it is the longest match that wins, not the first match.
4. The virtual `[Regular-Expression]` section. Within this section, it is the first match that wins.
5. The `default_filter_properties` statements.

Within the above order of evaluation, the first statement that matches wins, or determines how the transaction is handled. At most, one statement is executed. The filter file policy is executed by both proxy and cache transactions, so it is as if the filter file represented a single CPL `<Cache>` layer.

Appendix F: *Upgrading from CacheOS*

When upgrading from CacheOS version 4.*x* to the Proxy*SG*, the default policy files are created as follows:

- The CacheOS 4.*x* central filter file is copied to the Proxy*SG* central policy file with no changes.
- The CacheOS 4.*x* local filter file is copied to the Proxy*SG* local policy file with no changes.
- In addition, parts of the CacheOS 4.*x* security configuration are translated into CPL rules that are placed into the Visual Policy Manager (VPM) policy file.

When downgrading from Proxy*SG* to CacheOS 4.*x*, the system reverts to the most recent version of the configuration that was in effect before you upgraded. This includes any filter files that were used before the upgrade.

Using Backward-Compatibility Mode

The Content Policy Language (CPL) is almost completely backward compatible with the filter file language used in CacheOS version 4.*x*. This means that a CacheOS 4.*x* filter file can be used in the place of a policy file, and it will work, with a few differences. This is known as backward-compatibility mode. Before putting the Proxy*SG* into production, decide whether to continue to use the copied CacheOS 4.*x* filter files and run in backward-compatibility mode or convert your files to use standard CPL syntax. This distinction is on a per-file basis; for example, your central file could use standard CPL syntax while your local file remains a filter-style file.

Consider that the CPL compiler processes files in two different ways, depending on whether the file has the structure and syntax of a CacheOS 4.*x* filter file or a standard policy file. For filter-style files, the filter lines are rewritten into appropriate sections, then the statements and sections are evaluated in a specific order that is not determined by their ordering within the file. The compiler is then operating in backward-compatibility mode. For standard CPL-style policy files, layer ordering is important, with later layers overriding earlier layers.

When using the copied CacheOS 4.*x* filter files in the place of standard policy files, consider the following differences:

- The filter-file-specific `version_control` property is not supported.
- In CacheOS 4.*x*, filter patterns are case-sensitive unless `case_insensitive=yes` is specified. When the CPL compiler in Proxy*SG* processes the file, filter patterns are case-insensitive, unless `case_insensitive=no` or `case_sensitive=yes` is specified.
- A CacheOS 4.*x* filter file containing a `default_filter_properties` statement in the middle of a list of filters is be interpreted correctly by CPL. CacheOS 4.*x* only supported the use of `default_filter_properties` at the beginning and end of the filter list.
- In CacheOS 4.*x*, a prefix or domain-suffix filter pattern with a missing URL scheme is interpreted as an HTTP URL pattern. When processed by the CPL compiler, the same filter pattern matches a URL with any URL scheme (HTTP, HTTPS, FTP, MMS, RTSP).
- In a CacheOS 4.*x* filter file, if there is more than one prefix or domain-suffix filter with the same URL pattern, then all but the last filter is ignored, even if the filters have different ACL conditions.

For the CPL compiler, the correct filter will be selected at run time based on the ACL if the filters are distinguished by having different ACL conditions.

Converting Filter-Style Files to CPL Syntax

When converting your filter-style files, do not insert snippets of CPL syntax to take advantage of the new policy features, while leaving the bulk of the file unchanged. CPL and the CacheOS 4.*x* filter language have different orders of evaluation. If you insert CPL syntax into a filter file, then the CPL compiler assumes it is processing a standard policy file and switches to using the CPL order of evaluation.

To avoid this problem, convert your filter-style file to CPL syntax before modifying it to take advantage of CPL features. Do this using the following procedure.

To convert a filter-style file to use CPL syntax:

1. Load the filter file into the Proxy*SG* as the local policy file.
2. Issue the `view policy` command using the Management Console, or issue the `show policy` CLI command. This displays all of the loaded policy in CPL syntax: from the central, local, and VPM policy files.
3. Copy the section of the output of the `show policy` command corresponding to the filter file that you loaded into the local policy file.

Index

A

<Admin> layers, understanding 37
access_log( ) property 154
access_server() property 155
action definition block 246
action part, filter file 305
action.action_label( ) property 156
actions
    append() 228
    argument syntax in 227
    conflicting 47
    delete() 229
    delete_matching() 230
    log_message() 232
    notify_email 233, 234
    redirect() 235
    rewrite(url, regex_pattern, replacement_url) 237
    set(header, string) 240
    transform 242
    virus_check() 244
active content transformer 248
admin.access= condition 53
administrator transactions
    understanding 33
adminstrator transactions
    33
advertisement property, filter file 301
Advertisement transactions 35
advertisement( ) property 157
ALL statements 302
allow property 158
always_verify property, filter file 301
always_verify( ) property 159
append() action 228
attribute.name= condition 54
authenticate( ) 55
authenticate( ) property 160
authenticate.force() property 162
authenticate.mode( ) property 163
authenticated= condition 56

B

backward-compatibility mode 307
bitrate= condition 57
bypass_cache( ) property 167

C

cache property, filter file 302
cache transactions 33, 271
    understanding 35
cache( ) property 168
case_insensitive property, filter file 302
category= condition 59
<Cache> layer, understanding 38
check_authorization( ) property 170
client.address= condition 60
client.protocl= condition 61
comments, understanding 21
condition definition block 252
condition evaluation 35
condition= condition 62
conditions
    pattern-expression in 49
    trigger in 49
    user.x509.issuer= 147
conditions, filter file 301
conflicting actions 47
console_access= condition 64
Content Policy Language, see CPL ix
content pull transactions 35
content_filter_override( ). *See* request_filter_service
    property
content_management= condition 66
converting files to CPL syntax 308
cookie_sensitive( ) property 172
CPL
    concepts 19
    language basics 21
        comments 21
        definitions 25
        layers 22, 24
        notes 22
        quoting 22
        referential integrity 26
        rules 21
        sections 24
        substitutions 27
    policy model, understanding 20
    transaction overview 19
    upgrade/downgrade issues 30

D

date= condition 67
day= condition 68
define acl definition block, filter file 303
define action definition block 246
define category definition 250
define condition definition block 252
define prefix condition definition block 257, 261
define server_url.domain condition name definition 258
define subnet definition block 260
definition blocks
  define action 246
  define condition 252
  define prefix condition 257, 261
  define subnet 260
  transform active_content 248
  transform url_rewrite 270
definitions
  anonymous, overview 26
  define category 250
  define server_url.domain condition name 258
  overview 25
delete() action 229
delete_matching() action 230
delete_on_abandonment( ) property 173
deny( ) property 174
deny.unauthorized( ) property 175
deprecated syntax
  acl. *See* client.address=
  content_admin=. See content_management
  define domain. See url.domain 254
  protocol=. See client.protocol=
  socks.destination_address=. See socks=
direct property, filter file 302
direct( ) property 176
document
  conventions x
  organization ix
  related Blue Coat documentation x
domain-suffix filtering, filter file 304
Domain-Suffix section, filter file 304
downgrading 307
dynamic_bypass( ) property 177

E

earliest available time 35
<Exception> layers, understanding 39
evaluation order
  layers in CacheOS 4.x filter files 307
  layers in standard CPL-style policy files 307
  policy layers 307
evaluation order, filter file 306
exception( ) property 178
exception.autopad( ) property 179
exception.id=condition 69
exceptions
  layer ordering 46
  policies, using in 45
  using rule order 45

F

<Forward> layers, understanding 39
filter file
  acl definition block 303
  action part 305
  conditions 301
  default_filter_properties 305
  Domain-Suffix section 304
  evaluation order 306
  filter line 300
  filter part 300
  overview 299
  Prefix section 303
  properties 301
  Regular-Expression section 304
  section example 304
  sections 303
  structure 299
  syntax 299
filter file, ALL statements 302
filter line, filter file 300
force_cache( ) property 180
force_deny( ) property 181
force_exception( ) property 182
force_patience_page( ) property 183
forward 185
forward( ) property 184
forward.fail_open( ) property 185
forwarding transactions, understanding 36
ftp.method= condition 71
ftp.server_connection( ) property 186
ftp.server_data( ) propert 187
ftp_transport( ) property 188

G

group= condition 72

H

has_attribute.name= condition 74
has_client= condition 76
hour= condition 77
HTTP cache transactions 36
http.method= condition 79
http.request.version( ) property 190
http.request.version=condition 80
http.response.code=condition 81
http.response.version( ) property 191
http.response.version=condition 82
http.transparent_authentication=condition 83
http.x_method= condition 84

I

icp( ) property 192
If-modified-since transactions 35
im.buddy_id= condition 85
im.chat_room.conference= condition 86
im.chat_room.id= condition 87
im.chat_room.invite_only= condition 88
im.chat_room.member= condition 90
im.chat_room.type= condition 89
im.chat_room.voice_enabled= condition 91
im.file.extension= condition 92
im.file.name=condition 93
im.file.path= condition 94
im.file.size=condition 95
im.message.opcode= condition 96
im.message.route= condition 97
im.message.size= condition 98
im.message.text.contains= condition 99
im.message.type= condition 100
im.method=condition 101
im.strip_attachments( ) property 193
im.user_id= condition 102
integrate_new_hosts( ) property 194
Internet Explorer x
IP address 204, 260
IP subnet 260

J

Java Runtime Environment x
JRE x

L

label property, filter file 302
label( ) property. See action.action_label( ) property
label() property 195
latest commit time 35
layer guards, understanding 40
layers
  <Admin>, understanding 37
  <Cache> 38
  <Exception> 39
  <Forward> 39
  <Proxy> 40
  layer guards, understanding 40
  overview 24
  timing 41
  understanding 22, 37
live= condition 103
log.rewrite.field-id( ) property 196
log.suppress.field-id( ) property 197
log_message() action 232

M

max_bitrate( ) property 198
method= condition
  discussion 104
  protocols accepted 104
Microsoft Internet Explorer x
minute= condition 106
month= condition 107
Mozilla x

N

Netscape Communicator x
network interface card (NIC) 110
never_refresh_before_expiry( ) property 199
never_serve_after_expiry( ) property 200
notify_email action 233, 234

P

patience_page( ) property 201
pattern-expression 49
pipeline transactions 35
pipeline( ) property 202
policies
  best practices 44, 48
  blacklists 45
  defining 44
  definitive, making 47
  exceptions 45
  exceptions, layer ordering for 46
  exceptions, rule order for 45
  general use characters 29
  policy tracing 275

rules, conflicting 47
statistics, example 276
testing 275
tips on writing 44
troubleshooting 275
whitelists 45
policy ix
authentication/denial, setting 28
installing, overview 29
troubleshooting, overview 30
writing, overview 27
policy model, understanding 20
policy rules
order 45
policy tracing
enabling 275
example 276
request tracing, enabling 276
<Proxy> layers, understanding 40
prefetch property, filter file 302
prefetch property. See pipeline( )
prefix definition block 257, 261
prefix filtering, filter file 303
Prefix section, filter file 303
properties
access_log( ) 154
access_server() 155
action.action_label( ) 156
advertisement( ) 157
allow 158
always_verify( ) 159
authenticate( ) 160
authenticate.force() 162
authenticate.mode( ) 163
bypass_cache( ) 167
cache( ) 168
check_authorization( ) 170
cookie_sensitive( ) 172
delete_on_abandonment( ) 173
deny( ) 174
deny.unauthorized( ) 175
direct( ) 176
dynamic_bypass( ) 177
exception( ) 178
exception.autopad( ) 179
force_cache( ) 180
force_deny( ) 181
force_exception( ) 182
force_patience_page( ) 183
forward( ) 184
forward.fail_open( ) 185
ftp.server_connection( ) 186
ftp.server_data( ) 187
ftp_transport( ) 188
http.request.version( ) 190
http.response.version( ) 191
icp( ) 192
im.strip_attachments( ) 193
integrate_new_hosts( ) 194
label() 195
log.rewrite.field-id( ) 196
log.suppress.field-id( ) 197
max_bitrate( ) 198
never_refresh_before_expiry( ) 199
never_serve_after_expiry( ) 200
patience_page( ) 201
pipeline( ) 202
reflect_ip( ) 204
refresh( ) 206
remove_IMS_from_GET( ) 207
remove_PNC_from_GET( ) 208
remove_reload_from_IE_GET( ) 209
requesst.filter_service( ) 210
request.icap_service ( ) 212
response.icap_service ( ) 213
service() 214
socks.accelerate( ) 215
socks.authenticate( ) 216
socks.authenticate.force( ) 217
socks_gateway( ) 218
socks_gateway.fail_open( ) 219
streaming.transport( ) 220
trace.destination( ) 222
trace.request( ) 223
trace.rules() 224
ttl( ) 225
ua_sensitive( ) 226
properties, filter file 301
protocol= condition. *See* url.scheme
proxy transactions 33
policy evaluation 34
understanding 33
proxy.address= condition 109
proxy.card= condition 110
proxy.port= condition 111
proxy_authentication property, filter file 302
ProxySG, browsers supported ix

Q

quoting, understanding 22

R

realm= condition 112
redirect() action 235
references
  related Blue Coat documentation x
referential integrity, understanding 26
reflect_ip( ) property 204
reflect_vip( ) property. See reflect_ip( ) property
refresh property, filter file 302
refresh transactions 35
refresh( ) property 206
regular-expression filtering, filter file 304
Regular-Expression section, filter file 304
release.id= condition 114, 115
remove_IMS_from_GET( ) property 207
remove_PNC_from_GET( ) property 208
remove_reload_from_IE_GET( ) property 209
request.filter_service( ) property 210
request.header.address.header_name= condition 117
request.header.header_name= condition 116
request.icap_service ( ) property 212
request.tracing
  enabling 276
request.x_header.header_name= condition 121
request_header_address.header_name=. *See* request.header.header_name.address=
request_x_header_address.header_name= condition 122
response.header.header_name= condition 123
response.icap_service ( ) property 213
response.x_header.header_name= condition 124
rewrite(url, regex_pattern, replacement_url) action 237
[Rule] section, understanding 42
rule tracing
  enabling 275
  See<Default Parra Font> policy tracing
rules, overview 21

S

section guards, understanding 44
section types
  [Rule] 42
  [server_url.domain] 43
  [url.domain] 43
  [url.regex];[url.regex]
    section types, understanding 43
  [url] 43
  understanding 41
sections
  Domain-Suffix 304
  filter file example 304
  overview 24
  Prefix 303
  Regular-Expression 304
  section guards, understanding 44
  section types 41
  understanding 41
sections, filter file 303
[server_url.domain], section types, understanding 43
server_url.address condition 118, 125
server_url.domain= condition 118, 125
server_url.extension= condition 118, 125
server_url.host= condition 118, 125
server_url.path.regex= condition 118, 125
server_url.path= condition 118, 125
server_url.port= condition 118, 125
server_url.query.regex= condition 118, 125
server_url.regex= condition 118, 125
server_url.scheme= condition 118, 125
server_url= condition 118, 125
service property, filter file 302
service( ) property. See allow, deny( ), and exception( ) properties.
service() property 214
set(header, string) action 240
socks.accelerate( ) property 215
socks.accelerated= condition 129
socks.authenticate( ) property 216
socks.authenticate.force( ) property 217
socks.method=condition 129, 130
socks.version= condition 131
socks= condition 128
socks_gateway ( ) property 218
socks_gateway.fail_open( ) property 219
streaming
  troubleshoooting 220
  troubleshooting 167
streaming.client= condition 132
streaming.content= condition 133
streaming.transport( ) property 220
subnet 260
subnet definition block 260
substitutions, overview 27

T

time= condition 134
timing
  in layers, understanding 41
  understanding 36
trace.destination( ) 276
trace.destination( ) property 222
trace.request( ) property 223
trace.rules
  enabling 275
trace.rules() property 224
trace.rules, enabling. 275
transactions
  administrator 33
  cache 33, 35, 271
  forwarding 36
  overview 19
  proxy 33
  timing, understanding 36
  understanding 33
transform action 242
transform active_content definition block 248
transform url_rewrite definition block 270
trigger 49
troubleshooting
  can't use transport mechanism 220
  degraded performance 275
  Novell servers 74
  object isn't deleted 225
  policy tracing, using 275
  streaming 167
  telnet sessions 64
troubleshooting policies 275
ttl( ) property 225
tunneled= condition 136

U

ua_sensitive( ) property 226
upgrade/downgrade issues
  conditional compilation 31
  CPL syntax deprecations 30
  understanding 30
upgrading 307
[url], section type, understanding 43
URL rewrite transformer 270
url.address= condition 137
[url.domain], section types, understanding 43
url.domain= condition 137
url.extension= condition 137
url.host.is_numeric= condition 137
url.host.no_name= condition 137
url.host.regex= condition 137
url.host= condition 137
url.path.regex= condition 137
url.path= condition 137
url.port= condition 137
url.query.regex= condition 137
url.regex= condition 137
url= condition 137
user.domain= condition 146
user.x509.issuer= condition 147
user.x509.serialNumber= condition 148
user.x509.subject= condition 149
user= condition 144
Using CPL ix

V

virtual IP address 204
virus_check() action 244

W

weekday= condition 150

Y

year= condition 151