

[MS-FASP]: Firewall and Advanced Security Protocol Specification

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **Copyrights.** This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.
- **Patents.** Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft's Open Specification Promise (available here: <http://www.microsoft.com/interop/osp>) or the Community Promise (available here: <http://www.microsoft.com/interop/cp/default.aspx>). If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks.** The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights.
- **Fictitious Names.** The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications do not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them. Certain Open Specifications are intended for use in conjunction with publicly available standard specifications and network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

Revision Summary

Date	Revision History	Revision Class	Comments
04/03/2007	0.01		MCCP Milestone Longhorn Initial Availability
07/03/2007	1.0	Major	MLonghorn+90
07/20/2007	1.0.1	Editorial	Revised and edited the technical content.
08/10/2007	1.0.2	Editorial	Revised and edited the technical content.
09/28/2007	1.0.3	Editorial	Revised and edited the technical content.
10/23/2007	1.0.4	Editorial	Revised and edited the technical content.
11/30/2007	1.1	Minor	Updated the technical content.
01/25/2008	1.1.1	Editorial	Revised and edited the technical content.
03/14/2008	1.2	Minor	Updated the technical content.
05/16/2008	2.0	Major	Updated and revised the technical content.
06/20/2008	2.1	Minor	Updated the technical content.
07/25/2008	3.0	Major	Updated and revised the technical content.
08/29/2008	4.0	Major	Updated and revised the technical content.
10/24/2008	4.0.1	Editorial	Revised and edited the technical content.
12/05/2008	5.0	Major	Updated and revised the technical content.
01/16/2009	6.0	Major	Updated and revised the technical content.
02/27/2009	7.0	Major	Updated and revised the technical content.
04/10/2009	7.0.1	Editorial	Revised and edited the technical content.
05/22/2009	8.0	Major	Updated and revised the technical content.
07/02/2009	8.0.1	Editorial	Revised and edited the technical content.
08/14/2009	8.1	Minor	Updated the technical content.
09/25/2009	8.2	Minor	Updated the technical content.
11/06/2009	9.0	Major	Updated and revised the technical content.
12/18/2009	9.0.1	Editorial	Revised and edited the technical content.
01/29/2010	9.1	Minor	Updated the technical content.
03/12/2010	9.2	Minor	Updated the technical content.

Date	Revision History	Revision Class	Comments
04/23/2010	10.0	Major	Updated and revised the technical content.
06/04/2010	11.0	Major	Updated and revised the technical content.
07/16/2010	11.0	No change	No changes to the meaning, language, or formatting of the technical content.
08/27/2010	11.0	No change	No changes to the meaning, language, or formatting of the technical content.
10/08/2010	11.1	Minor	Clarified the meaning of the technical content.
11/19/2010	11.2	Minor	Clarified the meaning of the technical content.
01/07/2011	11.3	Minor	Clarified the meaning of the technical content.
02/11/2011	12.0	Major	Significantly changed the technical content.

Contents

1 Introduction	8
1.1 Glossary	8
1.2 References.....	8
1.2.1 Normative References.....	8
1.2.2 Informative References	9
1.3 Overview	10
1.4 Relationship to Other Protocols.....	11
1.5 Prerequisites/Preconditions	11
1.6 Applicability Statement.....	12
1.7 Versioning and Capability Negotiation.....	12
1.8 Vendor-Extensible Fields.....	12
1.9 Standards Assignments	13
2 Messages	14
2.1 Transport.....	14
2.2 Common Data Types	14
2.2.1 FW_STORE_TYPE	14
2.2.2 FW_PROFILE_TYPE	15
2.2.3 FW_POLICY_ACCESS_RIGHT	16
2.2.4 FW_IPV4_SUBNET	16
2.2.5 FW_IPV4_SUBNET_LIST	17
2.2.6 FW_IPV6_SUBNET	17
2.2.7 FW_IPV6_SUBNET_LIST	17
2.2.8 FW_IPV4_ADDRESS_RANGE	18
2.2.9 FW_IPV4_RANGE_LIST	18
2.2.10 FW_IPV6_ADDRESS_RANGE	18
2.2.11 FW_IPV6_RANGE_LIST.....	19
2.2.12 FW_PORT_RANGE.....	19
2.2.13 FW_PORT_RANGE_LIST.....	20
2.2.14 FW_PORT_KEYWORD	20
2.2.15 FW_PORTS	21
2.2.16 FW_ICMP_TYPE_CODE	21
2.2.17 FW_ICMP_TYPE_CODE_LIST	21
2.2.18 FW_INTERFACE_LUIDS	22
2.2.19 FW_DIRECTION.....	22
2.2.20 FW_INTERFACE_TYPE	23
2.2.21 FW_ADDRESS_KEYWORD	23
2.2.22 FW_ADDRESSES	24
2.2.23 FW_RULE_STATUS.....	25
2.2.24 FW_RULE_STATUS_CLASS.....	35
2.2.25 FW_OBJECT_CTRL_FLAG	36
2.2.26 FW_ENFORCEMENT_STATE	36
2.2.27 FW_OBJECT_METADATA	38
2.2.28 FW_OS_PLATFORM_OP	39
2.2.29 FW_OS_PLATFORM	40
2.2.30 FW_OS_PLATFORM_LIST	40
2.2.31 FW_RULE_ORIGIN_TYPE	40
2.2.32 FW_ENUM_RULES_FLAGS.....	41
2.2.33 FW_RULE_ACTION.....	42
2.2.34 FW_RULE_FLAGS.....	42

2.2.35	FW_RULE2_0	44
2.2.36	FW_RULE	45
2.2.37	FW_PROFILE_CONFIG	49
2.2.38	FW_GLOBAL_CONFIG_IPSEC_EXEMPT_VALUES	51
2.2.39	FW_GLOBAL_CONFIG_PRESHARED_KEY_ENCODING_VALUES	52
2.2.40	FW_GLOBAL_CONFIG_IPSEC_THROUGH_NAT_VALUES	53
2.2.41	FW_GLOBAL_CONFIG	53
2.2.42	FW_CONFIG_FLAGS	55
2.2.43	FW_NETWORK	56
2.2.44	FW_ADAPTER	56
2.2.45	FW_DIAG_APP	56
2.2.46	FW_RULE_CATEGORY	56
2.2.47	FW_PRODUCT	57
2.2.48	FW_IP_VERSION	58
2.2.49	FW_IPSEC_PHASE	58
2.2.50	FW_CS_RULE_FLAGS	58
2.2.51	FW_CS_RULE_ACTION	59
2.2.52	FW_CS_RULE2_0	60
2.2.53	FW_CS_RULE	61
2.2.54	FW_AUTH_METHOD	64
2.2.55	FW_AUTH_SUITE_FLAGS	65
2.2.56	FW_AUTH_SUITE	66
2.2.57	FW_AUTH_SET	67
2.2.58	FW_CRYPTO_KEY_EXCHANGE_TYPE	70
2.2.59	FW_CRYPTO_ENCRYPTION_TYPE	71
2.2.60	FW_CRYPTO_HASH_TYPE	72
2.2.61	FW_CRYPTO_PROTOCOL_TYPE	73
2.2.62	FW_PHASE1_CRYPTOSUITE	73
2.2.63	FW_PHASE2_CRYPTOSUITE	74
2.2.64	FW_PHASE1_CRYPTOFLAGS	75
2.2.65	FW_PHASE2_CRYPTOPFS	75
2.2.66	FW_CRYPTO_SET	76
2.2.67	FW_BYTE_BLOB	80
2.2.68	FW_COOKIE_PAIR	80
2.2.69	FW_PHASE1_KEY_MODULE_TYPE	80
2.2.70	FW_CERT_INFO	81
2.2.71	FW_AUTH_INFO	81
2.2.72	FW_ENDPOINTS	82
2.2.73	FW_PHASE1_SA_DETAILS	82
2.2.74	FW_PHASE2_TRAFFIC_TYPE	83
2.2.75	FW_PHASE2_SA_DETAILS	84
2.2.76	FW_PROFILE_CONFIG_VALUE	85
2.2.77	FW_MM_RULE	86
2.2.78	FW_CONN_HANDLE	88
2.2.79	FW_MATCH_KEY	88
2.2.80	FW_DATA_TYPE	89
2.2.81	FW_MATCH_VALUE	89
2.2.82	FW_MATCH_TYPE	90
2.2.83	FW_QUERY_CONDITION	90
2.2.84	FW_QUERY_CONDITIONS	91
2.2.85	FW_QUERY	92
2.2.86	FW_POLICY_STORE_HANDLE	93
2.2.87	FW_PRODUCT_HANDLE	93

3 Protocol Details	94
3.1 Server Details	94
3.1.1 Abstract Data Model	94
3.1.2 Timers	95
3.1.3 Initialization	95
3.1.4 Message Processing Events and Sequencing Rules	96
3.1.4.1 RRPC_FWOpenPolicyStore (Opnum 0)	102
3.1.4.2 RRPC_FWClosePolicyStore (Opnum 1)	103
3.1.4.3 RRPC_FWRestoreDefaults (Opnum 2)	104
3.1.4.4 RRPC_FWGetGlobalConfig (Opnum 3)	104
3.1.4.5 RRPC_FWSetGlobalConfig (Opnum 4)	106
3.1.4.6 RRPC_FWAddFirewallRule (Opnum 5)	107
3.1.4.7 RRPC_FWSetFirewallRule (Opnum 6)	108
3.1.4.8 RRPC_FWDeleteFirewallRule (Opnum 7)	109
3.1.4.9 RRPC_FWDeleteAllFirewallRules (Opnum 8)	110
3.1.4.10 RRPC_FWEnumFirewallRules (Opnum 9)	111
3.1.4.11 RRPC_FWGetConfig (Opnum 10)	112
3.1.4.12 RRPC_FWSetConfig (Opnum 11)	113
3.1.4.13 RRPC_FWAddConnectionSecurityRule (Opnum 12)	115
3.1.4.14 RRPC_FWSetConnectionSecurityRule (Opnum 13)	116
3.1.4.15 RRPC_FWDeleteConnectionSecurityRule (Opnum 14)	117
3.1.4.16 RRPC_FWDeleteAllConnectionSecurityRules (Opnum 15)	118
3.1.4.17 RRPC_FWEnumConnectionSecurityRules (Opnum 16)	119
3.1.4.18 RRPC_FWAddAuthenticationSet (Opnum 17)	120
3.1.4.19 RRPC_FWSetAuthenticationSet (Opnum 18)	121
3.1.4.20 RRPC_FWDeleteAuthenticationSet (Opnum 19)	122
3.1.4.21 RRPC_FWDeleteAllAuthenticationSets (Opnum 20)	123
3.1.4.22 RRPC_FWEnumAuthenticationSets (Opnum 21)	124
3.1.4.23 RRPC_FWAddCryptoSet (Opnum 22)	126
3.1.4.24 RRPC_FWSetCryptoSet (Opnum 23)	127
3.1.4.25 RRPC_FWDeleteCryptoSet (Opnum 24)	128
3.1.4.26 RRPC_FWDeleteAllCryptoSets (Opnum 25)	129
3.1.4.27 RRPC_FWEnumCryptoSets (Opnum 26)	130
3.1.4.28 RRPC_FWEnumPhase1SAs (Opnum 27)	132
3.1.4.29 RRPC_FWEnumPhase2SAs (Opnum 28)	133
3.1.4.30 RRPC_FWDeletePhase1SAs (Opnum 29)	134
3.1.4.31 RRPC_FWDeletePhase2SAs (Opnum 30)	135
3.1.4.32 RRPC_FWEnumProducts (Opnum 31)	135
3.1.4.33 RRPC_FWAddMainModeRule (Opnum 32)	136
3.1.4.34 RRPC_FWSetMainModeRule (Opnum 33)	137
3.1.4.35 RRPC_FWDeleteMainModeRule (Opnum 34)	138
3.1.4.36 RRPC_FWDeleteAllMainModeRules (Opnum 35)	139
3.1.4.37 RRPC_FWEnumMainModeRules (Opnum 36)	140
3.1.4.38 RRPC_FWQueryFirewallRules (Opnum 37)	141
3.1.4.39 RRPC_FWQueryConnectionSecurityRules (Opnum 38)	142
3.1.4.40 RRPC_FWQueryMainModeRules (Opnum 39)	143
3.1.4.41 RRPC_FWQueryAuthenticationSets (Opnum 40)	144
3.1.4.42 RRPC_FWQueryCryptoSets (Opnum 41)	145
3.1.4.43 RRPC_FWEnumNetworks (Opnum 42)	146
3.1.4.44 RRPC_FWEnumAdapters (Opnum 43)	147
3.1.4.45 RRPC_FWGetGlobalConfig2_10 (Opnum 44)	148
3.1.4.46 RRPC_FWGetConfig2_10 (Opnum 45)	149
3.1.4.47 RRPC_FWAddFirewallRule2_10 (Opnum 46)	151

3.1.4.48	RRPC_FWSetFirewallRule2_10 (Opnum 47)	152
3.1.4.49	RRPC_FWEnumFirewallRules2_10 (Opnum 48)	153
3.1.4.50	RRPC_FWAddConnectionSecurityRule2_10 (Opnum 49)	154
3.1.4.51	RRPC_FWSetConnectionSecurityRule2_10 (Opnum 50)	155
3.1.4.52	RRPC_FWEnumConnectionSecurityRules2_10 (Opnum 51)	157
3.1.4.53	RRPC_FWAddAuthenticationSet2_10 (Opnum 52)	158
3.1.4.54	RRPC_FWSetAuthenticationSet2_10 (Opnum 53)	159
3.1.4.55	RRPC_FWEnumAuthenticationSets2_10 (Opnum 54)	160
3.1.4.56	RRPC_FWAddCryptoSet2_10 (Opnum 55)	161
3.1.4.57	RRPC_FWSetCryptoSet2_10 (Opnum 56)	162
3.1.4.58	RRPC_FWEnumCryptoSets2_10 (Opnum 57)	163
3.1.5	Timer Events	164
3.1.6	Other Local Events	164
3.2	Client Details	165
3.2.1	Abstract Data Model	165
3.2.2	Timers	165
3.2.3	Initialization	165
3.2.4	Message Processing Events and Sequencing Rules	165
3.2.5	Timer Events	165
3.2.6	Other Local Events	165
4	Protocol Examples	166
4.1	Opening a Policy Store	166
4.2	Adding a Firewall Rule	166
4.3	Enumerating the Firewall Rules	168
4.4	Closing a Policy Store Handle	168
5	Security	170
5.1	Security Considerations for Implementers	170
5.2	Index of Security Parameters	170
6	Appendix A: Full IDL	171
7	Appendix B: Product Behavior	205
8	Change Tracking	210
9	Index	220

1 Introduction

The Firewall and Advanced Security Protocol describes managing security policies on remote computers. The specific policies that this protocol manages are those of the firewall and advanced security components. The protocol allows the same functionality that is available locally; it can add, modify, delete, and enumerate policies. It can also enumerate **security associations** that can be generated between hosts after this policy is enforced.

1.1 Glossary

The following terms are defined in [\[MS-GLOS\]](#):

- authentication header (AH)**
- Authenticated IP (AuthIP)**
- certificate authority (CA)**
- certificate revocation lists (CRL)**
- dynamic endpoint**
- endpoint**
- globally unique identifier (GUID)**
- Group Policy**
- Group Policy object (GPO)**
- Interface Definition Language (IDL)**
- Internet Key Exchange (IKE)**
- Internet Protocol Security (IPsec)**
- locally unique identifier (LUID)**
- Network Data Representation (NDR)**
- opnum**
- perfect forward secrecy (PFS)**
- remote procedure call (RPC)**
- Rivest-Shamir-Adleman (RSA)**
- RPC protocol sequence**
- RPC transport**
- security association (SA)**
- security identifier (SID)**
- Security Support Provider Interface (SSPI)**
- Transmission Control Protocol (TCP)**
- Transport Layer Security (TLS)**
- universally unique identifier (UUID)**

The following terms are specific to this document:

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [\[RFC2119\]](#). All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site,

<http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624>, as an additional source.

[C706] The Open Group, "DCE 1.1: Remote Procedure Call", C706, August 1997, <http://www.opengroup.org/public/pubs/catalog/c706.htm>

[MS-AIPS] Microsoft Corporation, "[Authenticated Internet Protocol Specification](#)", January 2007.

[MS-DTYP] Microsoft Corporation, "[Windows Data Types](#)", January 2007.

[MS-ERREF] Microsoft Corporation, "[Windows Error Codes](#)", January 2007.

[MS-IKEE] Microsoft Corporation, "[Internet Key Exchange Protocol Extensions](#)", January 2007.

[MS-IPHTTPS] Microsoft Corporation, "[IP over HTTPS \(IP-HTTPS\) Tunneling Protocol Specification](#)", December 2008.

[MS-KILE] Microsoft Corporation, "[Kerberos Protocol Extensions](#)", January 2007.

[MS-NLMP] Microsoft Corporation, "[NT LAN Manager \(NTLM\) Authentication Protocol Specification](#)", June 2007.

[MS-RPCE] Microsoft Corporation, "[Remote Procedure Call Protocol Extensions](#)", January 2007.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, <http://www.ietf.org/rfc/rfc2119.txt>

1.2.2 Informative References

[IANA-PROTO-NUM] Internet Assigned Numbers Authority, "Protocol Numbers", February 2007, <http://www.iana.org/assignments/protocol-numbers>

[IETF-TEREDO-EXT] Internet Engineering Task Force (IETF), "Teredo Extensions", <http://tools.ietf.org/html/draft-thaler-v6ops-teredo-extensions-05>

[IETF-TEREDO-UPDTS] Internet Engineering Task Force (IETF), "Teredo Security Updates", <http://tools.ietf.org/html/draft-krishnan-v6ops-teredo-update-05>

[MS-GLOS] Microsoft Corporation, "[Windows Protocols Master Glossary](#)", March 2007.

[MS-GPREG] Microsoft Corporation, "[Group Policy: Registry Extension Encoding](#)", August 2007.

[MSDN-C++NUMS/OPERS] Microsoft Corporation, "C++ Numbers and Operators", [http://msdn.microsoft.com/en-us/library/ff540372\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ff540372(VS.85).aspx)

[MSDN-IPV6PROTLEVEL] Microsoft Corporation, "Using IPV6_PROTECTION_LEVEL", <http://msdn.microsoft.com/en-us/library/aa916826.aspx>

[MSDN-OSVERSIONINFOEX] Microsoft Corporation, "OSVERSIONINFOEX Structure", <http://msdn.microsoft.com/en-us/library/ms724833.aspx>

[MSKB-935807] Microsoft Corporation, "Security Update for Windows Vista (KB935807)", August 2007, <http://www.microsoft.com/downloads/details.aspx?FamilyId=e9b64746-6afa-4a30-833d-e058e000c821&displaylang=en>

[MSWFPSDK] Microsoft Corporation, "Windows Filtering Platform", <http://msdn.microsoft.com/en-us/library/aa366510.aspx>

[RFC2409] Harkins, D., and Carrel, D., "The Internet Key Exchange (IKE)", RFC 2409, November 1998, <http://www.ietf.org/rfc/rfc2409.txt>

[RFC4301] Kent, S., and Seo, K., "Security Architecture for the Internet Protocol", RFC 4301, December 2005, <http://www.ietf.org/rfc/rfc4301.txt>

[RFC4380] Huitema, C., "Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)", RFC 4380, February 2006, <http://www.ietf.org/rfc/rfc4380.txt>

1.3 Overview

A host firewall is a software component that runs on host computers. It provides a layer of defense that can add depth to the collection of security measures when combined with other security measures, such as edge firewalls. Any threats that manage to get through the edge firewall, or those that are launched from within a corporate network, can still be defended against when host firewalls are used. Host firewalls are also useful in consumer scenarios in which there is, typically, no edge firewall to protect the home network.

Internet Protocol Security (IPsec) is a host-based, policy-driven security solution for protecting the host from all network access. IPsec focuses on connection security, which includes authentication, integrity protection, and confidentiality (encryption) of communication.

Because both IPsec and firewalls are host-based policy security technologies that operate in the network stack, they are managed together to avoid conflicts. Furthermore, firewall and connection security (IPsec) can interact, providing deeper and more effective filtering capabilities based on identities that are negotiated by IPsec as well as other IPsec state information. This document refers to this combined security solution as the firewall and advanced security components.

Firewall and advanced security components can be governed by policy that is received from local users or from network-wide policy that is distributed by an administrator, or both. There is a need in managed environments for a network administrator to be able to monitor the policies in effect on hosts, assuming that hosts may have received policies from both sources.

Network-wide policies are usually distributed by using **Group Policy objects (GPOs)** that live on active directories of domains. However, some workgroups or networks might not have a domain infrastructure. Even in non-domain joined environments, the network administrator needs to be able to remotely manage the advanced firewall and IPsec policy of a host.

Lastly, the network administrator might also be required to diagnose problems on the remote hosts. A common technique is to create temporary changes and then see if the changes fix the problem. This is the third scenario that warrants the capability to remotely administer host policies.

The Firewall and Advanced Security Protocol is designed and used to address the three needs previously mentioned. That is, its purpose is to monitor and manage remote host policies. It can manage all the policies that an administrator can manage locally. It can also monitor the specific policies coming from the different sources or monitor them aggregated, that is, all together, to understand and predict expected behavior. Lastly, it can make temporary modifications on the remote host policy to test online fixes and see whether they are effective.

The Firewall and Advanced Security Protocol is a client/server, **RPC**-based protocol. It consists of data types and methods. The data types are used to represent the different types of policy components that compose policy objects and policy configuration options. The methods are operations that are used to read and manage the different available policies. Therefore, the user can make method calls that pass new policy objects to be added to the policy, delete from the policy, or modify an existing object within the policy. The user can also call methods to retrieve all the policy

objects of interest. The following illustration shows read and write operations and their message sequences.

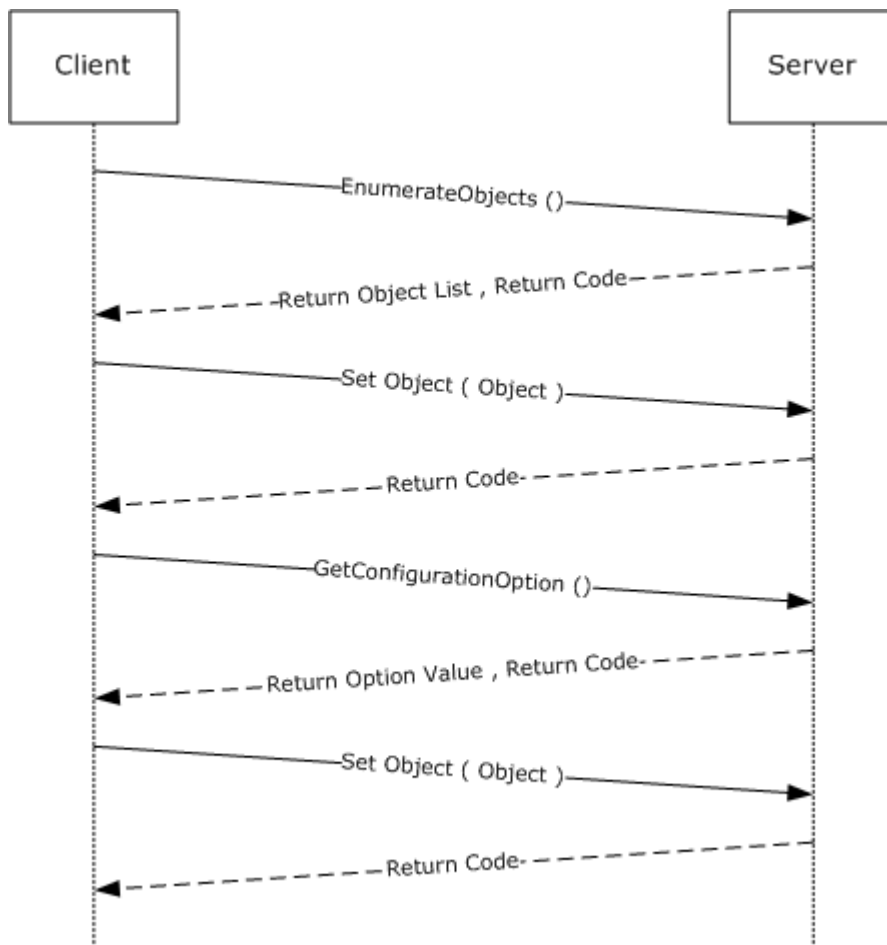


Figure 1: Read and write operations and their message sequences

The server role is represented by the host firewall, which contains the policy and enforces it. The client role is represented by the management console (or other user management tool), which sends, retrieves, and modifies the policies on the remote host firewall.

1.4 Relationship to Other Protocols

This protocol is implemented on RPC, as specified in [\[MS-RPCE\]](#), which uses the **Transmission Control Protocol (TCP)** as a transport. Aside from managing the policy for the firewall itself, this protocol is used to remotely manage the security policy database of the Security Architecture for the Internet Protocol [\[RFC4301\]](#), which describes how Internet Protocol Security (IPsec) should be enforced and what options the **Internet Key Exchange (IKE)** [\[RFC2409\]](#) and **Authenticated IP (AuthIP)** [\[MS-AIPS\]](#) have available to negotiate.

1.5 Prerequisites/Preconditions

This protocol assumes that the firewall and advanced security components have been initialized, are running, and have registered the corresponding RPC interface that is defined in section [2.1](#). This

protocol also assumes that the policy in the host firewall and advanced security components, which live in the server side, already allow the inbound traffic that the client computer, which is running the management tool, sends to the server during this protocol.

This protocol requires **Security Support Provider Interface (SSPI)** security by using packet privacy protection level (RPC_C_PROTECT_LEVEL_PKT_PRIVACY) and GSS negotiate authentication (RPC_C_AUTHN_GSS_NEGOTIATE), which negotiates between Kerberos Protocol Extensions [\[MS-KILE\]](#) and NT LAN Manager (NTLM) Authentication Protocol [\[MS-NLMP\]](#) authentication.

Lastly, the user on the client computer, which is running the management tool, must be able to impersonate an administrator or a network operator on the server-side computer.

1.6 Applicability Statement

This protocol should be used to address the needs defined in section [1.3](#).

1.7 Versioning and Capability Negotiation

This document covers versioning and capability negotiation issues in the following areas:

- Supported Transports: This protocol uses a single **RPC protocol sequence**, as specified in section [2.1](#).
- Protocol Versions: This protocol has only one interface version. There are currently three policy versions, which can be tied to policies and specific policy objects, as defined in section [2.2](#). The three versions are 0x0200, 0x0201, and 0x020A. [<1>](#) Protocol Versions are used as Binary Versions and Schema Versions (also called policy versions).
- Security and Authentication Methods: This protocol supports both Kerberos Protocol Extensions [\[MS-KILE\]](#) and NT LAN Manager (NTLM) Authentication Protocol [\[MS-NLMP\]](#) authentication methods, section [2.1](#).
- Localization: This protocol passes text strings without considering localization. However, some strings can be formatted in such a way that the firewall component knows where to look for localized versions of these strings, as defined in section [2.2](#). These strings can also be resolved with specific flags and method calls, as defined in section [3.1.4](#).
- Capability Negotiation: A configuration option defined in section [2.2.41](#) contains the maximum policy version and the binary supported by the server. With this option, a client can understand what can and cannot be expressed in this protocol and the methods that are supported to do so. The data types in section [2.2](#) and the existence and behavior of methods in section [3.1.4](#) are defined in terms of these policy versions when appropriate. No other negotiation capabilities, version-specific or otherwise, are present in this protocol.
- Byte order: All values defined in this specification are independent of whether the platform uses big-endian or little-endian byte order. For instance, protocol version 0x0200 = 512 decimal, and will be value 512 (0x0200) on both little-endian and big-endian platforms. Marshaling any values defined within this specification is handled by RPC (see [\[MS-RPCE\]](#)).

1.8 Vendor-Extensible Fields

This protocol uses Win32 error codes. These values are taken from the Microsoft Windows® error number space that is specified in [\[MS-ERREF\]](#). Vendors SHOULD reuse those values with their indicated meaning. Choosing any other value runs the risk of a collision in the future.

This protocol uses NTSTATUS values, as specified in [MS-ERREF]. Vendors can choose their own values for this field provided that the C bit (0x20000000) is set, indicating that it is a customer code.

Currently, vendors are not expected to extend this protocol. Therefore, the protocol does not consider provisions for extensions by parties other than Microsoft®.

1.9 Standards Assignments

Parameter	Value	Reference
RPC interface UUID for the Firewall and Advanced Security Protocol	6b5bdd1e-528c-422c-af8c-a4079be4fe48	Section 2.1

No standards assignments have been received for this protocol. All values used in these extensions are in private ranges specified in section [2.1](#). This protocol uses RPC **dynamic endpoints**, as specified in [C706] chapters [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), and [14](#).

2 Messages

2.1 Transport

This protocol uses the Remote Procedure Call (RPC) over TCP. It also uses RPC dynamic endpoints, as specified in [\[C706\]](#) chapters [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), and [14](#).

This RPC protocol MUST use Security Support Provider Interface (SSPI) security by using packet privacy protection level (RPC_C_PROTECT_LEVEL_PKT_PRIVACY) and GSS negotiate authentication (RPC_C_AUTHN_GSS_NEGOTIATE), which negotiates between Kerberos Protocol Extensions, as specified in [\[MS-KILE\]](#), and NT LAN Manager (NTLM) Authentication Protocol, as specified in [\[MS-NLMP\]](#) authentication.

This protocol MUST use the following universal unique identifier (UUID) interface: 6b5bdd1e-528c-422c-af8c-a4079be4fe48. The version number is 1.0.

2.2 Common Data Types

In addition to RPC base types and definitions specified in [\[C706\]](#) and [\[MS-DTYP\]](#), additional data types are defined in the sections that follow.

2.2.1 FW_STORE_TYPE

This data type defines enumerations used to identify store types.

```
typedef enum _tag_FW_STORE_TYPE
{
    FW_STORE_TYPE_INVALID,
    FW_STORE_TYPE_GP_RSOP,
    FW_STORE_TYPE_LOCAL,
    FW_STORE_TYPE_NOT_USED_VALUE_3,
    FW_STORE_TYPE_NOT_USED_VALUE_4,
    FW_STORE_TYPE_DYNAMIC,
    FW_STORE_TYPE_GPO,
    FW_STORE_TYPE_DEFAULTS,
    FW_STORE_TYPE_MAX
} FW_STORE_TYPE;
```

FW_STORE_TYPE_INVALID: This value is invalid and MUST NOT be used. It is defined for simplicity in writing **IDL** definitions and code. This symbolic constant has a value of zero.

FW_STORE_TYPE_GP_RSOP: This value identifies the store that contains all the policies from the different Group Policy objects (GPOs) that contain the networkwide policy. This store is persisted in the registry. It is downloaded by the **Group Policy** component (for more information, see [\[MS-GPREG\]](#)) and read by the firewall and advanced security components; therefore, it is a read-only store. This symbolic constant has a value of 1.

FW_STORE_TYPE_LOCAL: This value identifies the store that contains the local host policy. This store is persisted in the registry by the firewall and advanced security components; therefore, it is a read/write store. This symbolic constant has a value of 2.

FW_STORE_TYPE_NOT_USED_VALUE_3: This store is currently not used over the wire. This symbolic constant has a value of 3.

FW_STORE_TYPE_NOT_USED_VALUE_4: This store is currently not used over the wire. This symbolic constant has a value of 4.

FW_STORE_TYPE_DYNAMIC: This value identifies the store that contains the effective policy, that is, the aggregated and merged policy from all policy sources. Policy objects can be added and modified on this store, but they are not persisted and will be lost the next time the firewall and advanced security components initialize. Policy objects on this store can be modified only if they were originally added to this store. This symbolic constant has a value of 5.

FW_STORE_TYPE_GPO: This value is not used on the wire. This symbolic constant has a value of 6.

FW_STORE_TYPE_DEFAULTS: This value identifies the store that contains the defaults that the host operating system had out-of-box. This store is persisted in the registry. It is written by the host operating system setup. It is read by the firewall and advanced security components when it is instructed to go back to the default out-of-box configuration; hence it is a read-only store. This symbolic constant has a value of 7.

FW_STORE_TYPE_MAX: This value and greater values are invalid and MUST NOT be used; hence it is a read-only store. This symbolic constant is defined for simplicity in writing IDL definitions and code. It has a value of 8.

2.2.2 FW_PROFILE_TYPE

This data type defines the enumerations that are used to identify profile types. The enumeration values are bitmasks. Implementations MUST support using a single bitmask value and MUST support a combination of bitmask values. Valid combinations of bitmask values are all possible combinations using FW_PROFILE_TYPE_DOMAIN, FW_PROFILE_TYPE_PRIVATE, FW_PROFILE_TYPE_PUBLIC, and FW_PROFILE_TYPE_ALL. A profile is a set of networks to which a firewall policy might apply.

```
typedef [v1_enum] enum _tag_FW_PROFILE_TYPE
{
    FW_PROFILE_TYPE_INVALID = 0x000,
    FW_PROFILE_TYPE_DOMAIN = 0x001,
    FW_PROFILE_TYPE_STANDARD = 0x002,
    FW_PROFILE_TYPE_PRIVATE = 0x002,
    FW_PROFILE_TYPE_PUBLIC = 0x004,
    FW_PROFILE_TYPE_ALL = 0x7FFFFFFF,
    FW_PROFILE_TYPE_CURRENT = 0x80000000,
    FW_PROFILE_TYPE_NONE = 0x80000001
} FW_PROFILE_TYPE;
```

FW_PROFILE_TYPE_INVALID: This value is invalid and MUST NOT be used. It is defined for simplicity in writing IDL definitions and code.

FW_PROFILE_TYPE_DOMAIN: This value represents the profile for networks that are connected to domains.

FW_PROFILE_TYPE_STANDARD: This value represents the standard profile for networks. These networks are classified as private by the administrators in the server host. The classification happens the first time the host connects to the network. Usually these networks are behind Network Address Translation (NAT) devices, routers, and other edge devices, and they are in a private location, such as a home or an office.

FW_PROFILE_TYPE_PRIVATE: This value represents the profile for private networks, which is represented by the same value as that used for FW_PROFILE_TYPE_STANDARD.

FW_PROFILE_TYPE_PUBLIC: This value represents the profile for public networks. These networks are classified as public by the administrators in the server host. The classification happens the first time the host connects to the network. Usually these networks are those at airports, coffee shops, and other public places where the peers in the network or the network administrator are not trusted.

FW_PROFILE_TYPE_ALL: This value represents all these network sets and any future network sets.

FW_PROFILE_TYPE_CURRENT: This value represents the current profiles to which the firewall and advanced security components determine the host is connected at the moment of the call. This value can be specified only in method calls, and it cannot be combined with other flags.

FW_PROFILE_TYPE_NONE: This value represents no profile and is invalid. It is defined for simplicity in writing IDL definitions and code. This and greater values MUST NOT be used.

2.2.3 FW_POLICY_ACCESS_RIGHT

This enumeration defines access rights for the policy elements that can be accessed using the Firewall and Advanced Security Protocol. The values are not bitmasks and SHOULD NOT be used in bitwise OR operations.

```
typedef enum _tag_FW_POLICY_ACCESS_RIGHT
{
    FW_POLICY_ACCESS_RIGHT_INVALID,
    FW_POLICY_ACCESS_RIGHT_READ,
    FW_POLICY_ACCESS_RIGHT_READ_WRITE,
    FW_POLICY_ACCESS_RIGHT_MAX
} FW_POLICY_ACCESS_RIGHT;
```

FW_POLICY_ACCESS_RIGHT_INVALID: This value is invalid and MUST NOT be used. It is defined for simplicity in writing IDL definitions and code. This symbolic constant has a value of zero.

FW_POLICY_ACCESS_RIGHT_READ: This value represents a read-only access right. This symbolic constant has a value of 1.

FW_POLICY_ACCESS_RIGHT_READ_WRITE: This value represents a read and write access right. This symbolic constant has a value of 2.

FW_POLICY_ACCESS_RIGHT_MAX: This value and greater values are invalid and MUST NOT be used. This symbolic constant is defined for simplicity in writing IDL definitions and code. It has a value of 3.

2.2.4 FW_IPV4_SUBNET

This structure defines IPv4 subnets. It is used in policy rules.

```
typedef struct _tag_FW_IPV4_SUBNET {
    unsigned long dwAddress;
    unsigned long dwSubNetMask;
} FW_IPV4_SUBNET;
```



```
*PFW_IPV4_SUBNET;
```

dwAddress: This field represents the IPv4 address.

dwSubNetMask: This field contains the subnet mask in host network order. If it contains ones, they MUST be contiguous and shifted to the most significant bits.

A **dwSubNetMask** of 0x00000000 is invalid. A subnet mask of 0xFFFFFFFF means that the subnet mask represents a single address.

2.2.5 FW_IPV4_SUBNET_LIST

This structure is used to contain a number of [FW_IPV4_SUBNET](#) elements.

```
typedef struct _tag_FW_IPV4_SUBNET_LIST {  
    [range(0, 1000)] unsigned long dwNumEntries;  
    [size_is(dwNumEntries)] PFW_IPV4_SUBNET pSubNets;  
} FW_IPV4_SUBNET_LIST,  
*PFW_IPV4_SUBNET_LIST;
```

dwNumEntries: This field specifies the number of subnets that the structure contains.

pSubNets: A pointer to an array of **FW_IPV4_SUBNET** elements. The number of elements is given by **dwNumEntries**.

2.2.6 FW_IPV6_SUBNET

This structure represents an IPv6 subnet.

```
typedef struct _tag_FW_IPV6_SUBNET {  
    unsigned char Address[16];  
    [range(0, 128)] unsigned long dwNumPrefixBits;  
} FW_IPV6_SUBNET,  
*PFW_IPV6_SUBNET;
```

Address: This field contains a 16-octet IPv6 address.

dwNumPrefixBits: This field contains the number of more-significant bits that represent the IPv6 subnet.

The **dwNumPrefixBits** MUST NOT be greater than 128 and not less than 1. The address SHOULD NOT be an unspecified address (an address composed of all zeros), [<2>](#) and it MUST not be a loopback address.

2.2.7 FW_IPV6_SUBNET_LIST

This structure is used to contain a number of [FW_IPV6_SUBNET](#) elements.

```
typedef struct _tag_FW_IPV6_SUBNET_LIST {  
    [range(0, 1000)] unsigned long dwNumEntries;
```

```

    [size_is(dwNumEntries)] PFW_IPV6_SUBNET pSubNets;
} FW_IPV6_SUBNET_LIST,
*PFW_IPV6_SUBNET_LIST;

```

dwNumEntries: This field specifies the number of subnets that the structure contains.

pSubNets: A pointer to an array of **FW_IPV6_SUBNET** elements. The number of elements is given by **dwNumEntries**.

2.2.8 FW_IPV4_ADDRESS_RANGE

This structure represents a range of IPv4 addresses within the IPv4 address space.

```

typedef struct _tag_FW_IPV4_ADDRESS_RANGE {
    unsigned long dwBegin;
    unsigned long dwEnd;
} FW_IPV4_ADDRESS_RANGE,
*PFW_IPV4_ADDRESS_RANGE;

```

dwBegin: The first IPv4 address of the range in the IPv4 address space defined by this structure. The address is included in the range.

dwEnd: The last IPv4 address of the range in the IPv4 address space defined by this structure. The address is included in the range.

Valid **FW_IPV4_ADDRESS_RANGE** structures MUST have a **dwBegin** value less than or equal to the **dwEnd** value. Structures with **dwBegin** equal to **dwEnd** represent a single IPv4 address.

2.2.9 FW_IPV4_RANGE_LIST

This structure is used to contain a number of [FW_IPV4_ADDRESS_RANGE](#) elements.

```

typedef struct _tag_FW_IPV4_RANGE_LIST {
    [range(0, 1000)] unsigned long dwNumEntries;
    [size_is(dwNumEntries)] PFW_IPV4_ADDRESS_RANGE pRanges;
} FW_IPV4_RANGE_LIST,
*PFW_IPV4_RANGE_LIST;

```

dwNumEntries: This field specifies the number of IPv4 address ranges that the structure contains.

pRanges: A pointer to an array of **FW_IPV4_ADDRESS_RANGE** elements. The number of elements is given by **dwNumEntries**.

2.2.10 FW_IPV6_ADDRESS_RANGE

This structure represents a range of IPv6 addresses within the IPv6 address space.

```

typedef struct _tag_FW_IPV6_ADDRESS_RANGE {
    unsigned char Begin[16];

```

```
    unsigned char End[16];
} FW_IPV6_ADDRESS_RANGE,
*PFW_IPV6_ADDRESS_RANGE;
```

Begin: A 16-octet array containing the first IPv6 address of the range in the IPv6 address range defined by this structure.

End: A 16-octet array containing the last IPv6 address of the range in the IPv6 address range defined by this structure.

Valid **FW_IPV6_ADDRESS_RANGE** structures MUST have a **Begin** value less than or equal to the **End** value. Structures with **Begin** equal to **End** represent a single IPv6 address. **Begin** and **End** MUST NOT contain either an unspecified or a loopback address.

Begin and **End** are in network order.

2.2.11 FW_IPV6_RANGE_LIST

This structure is used to contain a number of [FW_IPV6_ADDRESS_RANGE](#) elements.

```
typedef struct _tag_FW_IPV6_RANGE_LIST {
    [range(0, 1000)] unsigned long dwNumEntries;
    [size_is(dwNumEntries)] PFW_IPV6_ADDRESS_RANGE pRanges;
} FW_IPV6_RANGE_LIST,
*PFW_IPV6_RANGE_LIST;
```

dwNumEntries: This field specifies the number of IPv6 address ranges that the structure contains.

pRanges: A pointer to an array of **FW_IPV6_ADDRESS_RANGE** elements. The number of elements is given by **dwNumEntries**.

2.2.12 FW_PORT_RANGE

This structure represents a range of ports. Ports are 16-bit unsigned values used in TCP and UDP protocols.

```
typedef struct _tag_FW_PORT_RANGE {
    unsigned short wBegin;
    unsigned short wEnd;
} FW_PORT_RANGE,
*PFW_PORT_RANGE;
```

wBegin: This field specifies the first port included in the range defined.

wEnd: This field specifies the last port included in the range defined.

Valid **FW_PORT_RANGE** structures MUST have a **wBegin** value less than or equal to the **wEnd** value. In this protocol, **wBegin** is equal to **wEnd**.

2.2.13 FW_PORT_RANGE_LIST

This structure is used to contain a number of [FW_PORT_RANGE](#) elements.

```
typedef struct _tag_FW_PORT_RANGE_LIST {
    [range(0, 1000)] unsigned long dwNumEntries;
    [size_is(dwNumEntries)] PFW_PORT_RANGE pPorts;
} FW_PORT_RANGE_LIST,
*PFW_PORT_RANGE_LIST;
```

dwNumEntries: This field specifies the number of port ranges that the structure contains.

pPorts: A pointer to an array of [FW_PORT_RANGE](#) elements. The number of elements is given as **dwNumEntries**.

2.2.14 FW_PORT_KEYWORD

This enumeration identifies (with bitmask flags) the ports of specific well-known protocols. As specified in the following descriptions, these ports can change dynamically.

```
typedef enum _tag_FW_PORT_KEYWORD
{
    FW_PORT_KEYWORD_NONE = 0x00,
    FW_PORT_KEYWORD_DYNAMIC_RPC_PORTS = 0x01,
    FW_PORT_KEYWORD_RPC_EP = 0x02,
    FW_PORT_KEYWORD_TEREDO_PORT = 0x04,
    FW_PORT_KEYWORD_IP_TLS_IN = 0x08,
    FW_PORT_KEYWORD_IP_TLS_OUT = 0x10,
    FW_PORT_KEYWORD_MAX = 0x20,
    FW_PORT_KEYWORD_MAX_V2_1 = 0x08
} FW_PORT_KEYWORD;
```

FW_PORT_KEYWORD_NONE: Specifies that no port keywords are used.

FW_PORT_KEYWORD_DYNAMIC_RPC_PORTS: Represents ports that are used by RPC servers that are assigned dynamically at runtime. A port is represented by this keyword only while it is in use by such an RPC server. For information about dynamic RPC ports, see [\[MS-RPCE\]](#).

FW_PORT_KEYWORD_RPC_EP: Represents ports that are used by an RPC **endpoint** mapper only when an RPC interface is registered for the corresponding **RPC transport**. This keyword currently can represent only port 135 (for the RPC over TCP transport) and port 593 (for the RPC over HTTP transport).

FW_PORT_KEYWORD_TEREDO_PORT: Represents the random dynamic port that is used by the Teredo service only while the Teredo service is in use. The Teredo service described in this protocol implements Teredo Extensions. For more information, see [\[IETF-TEREDO-EXT\]](#), [\[IETF-TEREDO-UPDTS\]](#), and [\[RFC4380\]](#).

FW_PORT_KEYWORD_IP_TLS_IN: Represents the inbound TCP ports that are used by the IPTLS service (see IP-HTTPS server on [\[MS-IPHTTPS\]](#) section 1.1), but only while the IPTLS service is in use. The IPTLS service that is described in this protocol implements IP over **Transport Layer Security (TLS)**. For more information, see [\[MS-IPHTTPS\]](#).

FW_PORT_KEYWORD_IP_TLS_OUT: Represents the outbound TCP ports that are used by the IPTLS service (see IP-HTTPS server on [\[MS-IPHTTPS\]](#) section 1.1), but only while the IPTLS service is in use. The IPTLS service that is described in this protocol implements IP over TLS. For valid port values, see "IP-HTTPS server" in [\[MS-IPHTTPS\]](#) section *r*.

FW_PORT_KEYWORD_MAX: This is an invalid value. It is defined for simplicity in writing IDL definitions and code. This value and greater values MUST NOT be used.

FW_PORT_KEYWORD_MAX_V2_1: This value and greater values are invalid and MUST NOT be used by servers and clients with schema version 2.1 and earlier (see section [2.2.37](#)). This value is defined for simplicity in writing IDL definitions and code.

2.2.15 FW_PORTS

This structure contains the ports represented statically through [FW_PORT_RANGE](#) structures or symbolically through [FW_PORT_KEYWORD](#) enumeration values.

```
typedef struct _tag_FW_PORTS {
    unsigned short wPortKeywords;
    FW_PORT_RANGE_LIST Ports;
} FW_PORTS,
*PFW_PORTS;
```

wPortKeywords: This field is a combination of **FW_PORT_KEYWORDS**.

Ports: This field is a list of specifically defined ports.

2.2.16 FW_ICMP_TYPE_CODE

This data type defines ICMP (internet control message protocol with protocol numbers assigned in [\[IANA-PROTO-NUM\]](#)) message types and codes. It specifies an ICMP type and either its specific code or all codes for that type.

```
typedef struct _tag_FW_ICMP_TYPE_CODE {
    unsigned char bType;
    [range(0, 256)] unsigned short wCode;
} FW_ICMP_TYPE_CODE,
*PFW_ICMP_TYPE_CODE;
```

bType: This field specifies the ICMP type.

wCode: This field specifies the ICMP code.

The **wCode** field MUST contain values between 0x0000 and 0x0100. When **wCode** contains 0x100, it expresses any ICMP code belonging to the corresponding ICMP type. When **wCode** contains values in the range 0 to 0x00FF, it expresses a specific ICMP code.

All valid ICMP type and code combinations are valid, even those not currently assigned for a specific use.

2.2.17 FW_ICMP_TYPE_CODE_LIST

This structure is used to contain a number of [FW_ICMP_TYPE_CODE](#) elements.

```
typedef struct _tag_FW_ICMP_TYPE_CODE_LIST {
    [range(0, 1000)] unsigned long dwNumEntries;
    [size_is(dwNumEntries)] PFW_ICMP_TYPE_CODE pEntries;
} FW_ICMP_TYPE_CODE_LIST,
*PFW_ICMP_TYPE_CODE_LIST;
```

dwNumEntries: This field specifies the number of **FW_ICMP_TYPE_CODE** elements that the structure contains.

pEntries: A pointer to an array of **FW_ICMP_TYPE_CODE** elements. The number of elements is given by **dwNumEntries**.

2.2.18 FW_INTERFACE_LUIDS

This structure is used to contain **locally unique identifier (LUID)** values that uniquely represent single network adapters (NICs) within a specific computer.

```
typedef struct _tag_FW_INTERFACE_LUIDS {
    [range(0, 1000)] unsigned long dwNumLUIDs;
    [size_is(dwNumLUIDs)] GUID* pLUIDs;
} FW_INTERFACE_LUIDS,
*PFW_INTERFACE_LUIDS;
```

dwNumLUIDs: This field specifies the number of interface LUIDs that the structure contains.

pLUIDs: A pointer to an array of **GUID** elements. The number of elements is given by **dwNumLUIDs**. The **GUID** data type is specified in [\[MS-DTYP\]](#).

2.2.19 FW_DIRECTION

This enumeration represents the direction of network traffic flow.

```
typedef enum _tag_FW_DIRECTION
{
    FW_DIR_INVALID = 0,
    FW_DIR_IN,
    FW_DIR_OUT,
    FW_DIR_MAX
} FW_DIRECTION;
```

FW_DIR_INVALID: This is an invalid value, and it MUST NOT be used. It is defined for simplicity in writing IDL definitions and code. This symbolic constant has a value of zero.

FW_DIR_IN: Specifies an inbound network traffic flow. These are flows that are initiated by a remote machine toward the local machine. This symbolic constant has a value of 1.

FW_DIR_OUT: Specifies an outbound network traffic flow. These are flows that are initiated by the local machine toward a remote machine. This symbolic constant has a value of 2.

FW_DIR_MAX: This value is invalid and MUST NOT be used. This symbolic constant is defined for simplicity in writing IDL definitions and code. It has a value of 3.

2.2.20 FW_INTERFACE_TYPE

This enumeration is used to represent types of network adapters (NICs) in a specific machine. Each type might have one or more network adapters.

```
typedef enum _tag_FW_INTERFACE_TYPE
{
    FW_INTERFACE_TYPE_ALL = 0x0000,
    FW_INTERFACE_TYPE_LAN = 0x0001,
    FW_INTERFACE_TYPE_WIRELESS = 0x0002,
    FW_INTERFACE_TYPE_REMOTE_ACCESS = 0x0004,
    FW_INTERFACE_TYPE_MAX = 0x0008
} FW_INTERFACE_TYPE;
```

FW_INTERFACE_TYPE_ALL: Represents all types of network adapters (NICs). The following types fall into this type.

FW_INTERFACE_TYPE_LAN: Represents network adapters (NICs) that use wired network physical layers such as Ethernet.

FW_INTERFACE_TYPE_WIRELESS: Represents network adapters that use the wireless 802 network physical layer.

FW_INTERFACE_TYPE_REMOTE_ACCESS: Represents network adapters that use VPN connections.

FW_INTERFACE_TYPE_MAX: This value and greater values are invalid and MUST NOT be used. This value is defined for simplicity in writing IDL definitions and code.

2.2.21 FW_ADDRESS_KEYWORD

This enumeration is used to represent specific address types. As specified in the following descriptions, these address types can change dynamically.

```
typedef enum _tag_FW_ADDRESS_KEYWORD
{
    FW_ADDRESS_KEYWORD_NONE = 0x0000,
    FW_ADDRESS_KEYWORD_LOCAL_SUBNET = 0x0001,
    FW_ADDRESS_KEYWORD_DNS = 0x0002,
    FW_ADDRESS_KEYWORD_DHCP = 0x0004,
    FW_ADDRESS_KEYWORD_WINS = 0x0008,
    FW_ADDRESS_KEYWORD_DEFAULT_GATEWAY = 0x0010,
    FW_ADDRESS_KEYWORD_MAX = 0x0020
} FW_ADDRESS_KEYWORD;
```

FW_ADDRESS_KEYWORD_NONE: Specifies that no specific keyword is used.

FW_ADDRESS_KEYWORD_LOCAL_SUBNET: Represents the collection of addresses that are currently within the local subnet of the computer.

FW_ADDRESS_KEYWORD_DNS: Represents the collection of addresses of the current DNS servers.

FW_ADDRESS_KEYWORD_DHCP: Represents the collection of addresses of the current DHCP servers.

FW_ADDRESS_KEYWORD_WINS: Represents the collection of addresses of the current WINS servers.

FW_ADDRESS_KEYWORD_DEFAULT_GATEWAY: Represents the collection of addresses of the current gateway servers.

FW_ADDRESS_KEYWORD_MAX: This value and greater values are invalid and MUST NOT be used. This value is defined for simplicity in writing IDL definitions and code.

2.2.22 FW_ADDRESSES

This structure contains a list of address structures. Static and symbolic representations are supported, but a structure can contain only one representation type. The address structure representations follow:

Static Representation

- [FW_IPV4_SUBNET_LIST](#)
- [FW_IPV4_RANGE_LIST](#)
- [FW_IPV6_SUBNET_LIST](#)
- [FW_IPV6_RANGE_LIST](#)

Symbolic Representation

- [FW_ADDRESS_KEYWORD](#) enumeration values

The FW_ADDRESSES definition follows:

```
typedef struct _tag_FW_ADDRESSES {
    unsigned long dwV4AddressKeywords;
    unsigned long dwV6AddressKeywords;
    FW_IPV4_SUBNET_LIST V4SubNets;
    FW_IPV4_RANGE_LIST V4Ranges;
    FW_IPV6_SUBNET_LIST V6SubNets;
    FW_IPV6_RANGE_LIST V6Ranges;
} FW_ADDRESSES,
*PFW_ADDRESSES;
```

dwV4AddressKeywords: A combination of **FW_ADDRESS_KEYWORD** flags. Addresses in this field are specified from the IPv4 address space.

dwV6AddressKeywords: A combination of **FW_ADDRESS_KEYWORD** flags. Addresses in this field are specified from the IPv6 address space.

V4SubNets: A list of specifically defined IPv4 address subnets.

V4Ranges: A list of specifically defined IPv4 address ranges.

V6SubNets: A list of specifically defined IPv6 address subnets.

V6Ranges: A list of specifically defined IPv6 address ranges.

2.2.23 FW_RULE_STATUS

This enumeration represents status codes that identify the error states of a policy object, including successful states. If an object is in an erroneous state, the enumeration value represents a reason for the error.

```
typedef [v1_enum] enum _tag_FW_RULE_STATUS
{
    FW_RULE_STATUS_OK = 0x00010000,
    FW_RULE_STATUS_PARTIALLY_IGNORED = 0x00020000,
    FW_RULE_STATUS_IGNORED = 0x00040000,
    FW_RULE_STATUS_PARSING_ERROR_NAME = 0x00080001,
    FW_RULE_STATUS_PARSING_ERROR_DESC = 0x00080002,
    FW_RULE_STATUS_PARSING_ERROR_APP = 0x00080003,
    FW_RULE_STATUS_PARSING_ERROR_SVC = 0x00080004,
    FW_RULE_STATUS_PARSING_ERROR_RMA = 0x00080005,
    FW_RULE_STATUS_PARSING_ERROR_RUA = 0x00080006,
    FW_RULE_STATUS_PARSING_ERROR_EMBD = 0x00080007,
    FW_RULE_STATUS_PARSING_ERROR_RULE_ID = 0x00080008,
    FW_RULE_STATUS_PARSING_ERROR_PHASE1_AUTH = 0x00080009,
    FW_RULE_STATUS_PARSING_ERROR_PHASE2_CRYPT0 = 0x0008000A,
    FW_RULE_STATUS_PARSING_ERROR_PHASE2_AUTH = 0x0008000B,
    FW_RULE_STATUS_PARSING_ERROR_RESOLVE_APP = 0x0008000C,
    FW_RULE_STATUS_PARSING_ERROR_MAINMODE_ID = 0x0008000D,
    FW_RULE_STATUS_PARSING_ERROR_PHASE1_CRYPT0 = 0x0008000E,
    FW_RULE_STATUS_PARSING_ERROR = 0x00080000,
    FW_RULE_STATUS_SEMANTIC_ERROR_RULE_ID = 0x00100010,
    FW_RULE_STATUS_SEMANTIC_ERROR_PORTS = 0x00100020,
    FW_RULE_STATUS_SEMANTIC_ERROR_PORT_KEYW = 0x00100021,
    FW_RULE_STATUS_SEMANTIC_ERROR_PORT_RANGE = 0x00100022,
    FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_V4_SUBNETS = 0x00100040,
    FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_V6_SUBNETS = 0x00100041,
    FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_V4_RANGES = 0x00100042,
    FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_V6_RANGES = 0x00100043,
    FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_RANGE = 0x00100044,
    FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_MASK = 0x00100045,
    FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_PREFIX = 0x00100046,
    FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_KEYW = 0x00100047,
    FW_RULE_STATUS_SEMANTIC_ERROR_LADDR_PROP = 0x00100048,
    FW_RULE_STATUS_SEMANTIC_ERROR_RADDR_PROP = 0x00100049,
    FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_V6 = 0x0010004A,
    FW_RULE_STATUS_SEMANTIC_ERROR_LADDR_INTF = 0x0010004B,
    FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_V4 = 0x0010004C,
    FW_RULE_STATUS_SEMANTIC_ERROR_TUNNEL_ENDPOINT_ADDR = 0x0010004D,
    FW_RULE_STATUS_SEMANTIC_ERROR_DTE_VER = 0x0010004E,
    FW_RULE_STATUS_SEMANTIC_ERROR_DTE_MISMATCH_ADDR = 0x0010004F,
    FW_RULE_STATUS_SEMANTIC_ERROR_PROFILE = 0x00100050,
    FW_RULE_STATUS_SEMANTIC_ERROR_ICMP = 0x00100060,
    FW_RULE_STATUS_SEMANTIC_ERROR_ICMP_CODE = 0x00100061,
    FW_RULE_STATUS_SEMANTIC_ERROR_IF_ID = 0x00100070,
    FW_RULE_STATUS_SEMANTIC_ERROR_IF_TYPE = 0x00100071,
    FW_RULE_STATUS_SEMANTIC_ERROR_ACTION = 0x00100080,
    FW_RULE_STATUS_SEMANTIC_ERROR_ALLOW_BYPASS = 0x00100081,
    FW_RULE_STATUS_SEMANTIC_ERROR_DO_NOT_SECURE = 0x00100082,
    FW_RULE_STATUS_SEMANTIC_ERROR_ACTION_BLOCK_IS_ENCRYPTED_SECURE = 0x00100083,
    FW_RULE_STATUS_SEMANTIC_ERROR_DIR = 0x00100090,
    FW_RULE_STATUS_SEMANTIC_ERROR_PROT = 0x001000A0,
    FW_RULE_STATUS_SEMANTIC_ERROR_PROT_PROP = 0x001000A1,
    FW_RULE_STATUS_SEMANTIC_ERROR_DEFER_EDGE_PROP = 0x001000A2,
```

FW_RULE_STATUS_SEMANTIC_ERROR_ALLOW_BYPASS_OUTBOUND = 0x001000A3,
FW_RULE_STATUS_SEMANTIC_ERROR_DEFER_USER_INVALID_RULE = 0x001000A4,
FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS = 0x001000B0,
FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_AUTO_AUTH = 0x001000B1,
FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_AUTO_BLOCK = 0x001000B2,
FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_AUTO_DYN_RPC = 0x001000B3,
FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_AUTHENTICATE_ENCRYPT = 0x001000B4,
FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_AUTH_WITH_ENC_NEGOTIATE_VER = 0x001000B5,
FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_AUTH_WITH_ENC_NEGOTIATE = 0x001000B6,
FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_ESP_NO_ENCAP_VER = 0x001000B7,
FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_ESP_NO_ENCAP = 0x001000B8,
FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_TUNNEL_AUTH_MODES_VER = 0x001000B9,
FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_TUNNEL_AUTH_MODES = 0x001000BA,
FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_IP_TLS_VER = 0x001000BB,
FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_PORTRANGE_VER = 0x001000BC,
FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_ADDRS_TRAVERSE_DEFER_VER = 0x001000BD,
FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_AUTH_WITH_ENC_NEGOTIATE_OUTBOUND = 0x001000BE,
FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_AUTHENTICATE_WITH_OUTBOUND_BYPASS_VER = 0x001000BF,
FW_RULE_STATUS_SEMANTIC_ERROR_REMOTE_AUTH_LIST = 0x001000C0,
FW_RULE_STATUS_SEMANTIC_ERROR_REMOTE_USER_LIST = 0x001000C1,
FW_RULE_STATUS_SEMANTIC_ERROR_PLATFORM = 0x001000E0,
FW_RULE_STATUS_SEMANTIC_ERROR_PLATFORM_OP_VER = 0x001000E1,
FW_RULE_STATUS_SEMANTIC_ERROR_PLATFORM_OP = 0x001000E2,
FW_RULE_STATUS_SEMANTIC_ERROR_DTE_NOANY_ADDR = 0x001000F0,
FW_RULE_STATUS_SEMANTIC_TUNNEL_EXEMPT_WITH_GATEWAY = 0x001000F1,
FW_RULE_STATUS_SEMANTIC_TUNNEL_EXEMPT_VER = 0x001000F2,
FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_AUTH_SET_ID = 0x00100500,
FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPT0_SET_ID = 0x00100510,
FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPT0_SET_ID = 0x00100511,
FW_RULE_STATUS_SEMANTIC_ERROR_SET_ID = 0x00101000,
FW_RULE_STATUS_SEMANTIC_ERROR_IPSEC_PHASE = 0x00101010,
FW_RULE_STATUS_SEMANTIC_ERROR_EMPTY_SUITES = 0x00101020,
FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_AUTH_METHOD = 0x00101030,
FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_AUTH_METHOD = 0x00101031,
FW_RULE_STATUS_SEMANTIC_ERROR_AUTH_METHOD_ANONYMOUS = 0x00101032,
FW_RULE_STATUS_SEMANTIC_ERROR_AUTH_METHOD_DUPLICATE = 0x00101033,
FW_RULE_STATUS_SEMANTIC_ERROR_AUTH_SUITE_FLAGS = 0x00101040,
FW_RULE_STATUS_SEMANTIC_ERROR_HEALTH_CERT = 0x00101041,
FW_RULE_STATUS_SEMANTIC_ERROR_AUTH_SIGNCERT_VER = 0x00101042,
FW_RULE_STATUS_SEMANTIC_ERROR_AUTH_INTERMEDIATE_CA_VER = 0x00101043,
FW_RULE_STATUS_SEMANTIC_ERROR_MACHINE_SHKEY = 0x00101050,
FW_RULE_STATUS_SEMANTIC_ERROR_CA_NAME = 0x00101060,
FW_RULE_STATUS_SEMANTIC_ERROR_MIXED_CERTS = 0x00101061,
FW_RULE_STATUS_SEMANTIC_ERROR_NON_CONTIGUOUS_CERTS = 0x00101062,
FW_RULE_STATUS_SEMANTIC_ERROR_MIXED_CA_TYPE_IN_BLOCK = 0x00101063,
FW_RULE_STATUS_SEMANTIC_ERROR_MACHINE_USER_AUTH = 0x00101070,
FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPT0_NON_DEFAULT_ID = 0x00105000,
FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPT0_FLAGS = 0x00105001,
FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPT0_TIMEOUT_MINUTES = 0x00105002,
FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPT0_TIMEOUT_SESSIONS = 0x00105003,
FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPT0_KEY_EXCHANGE = 0x00105004,
FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPT0_ENCRYPTION = 0x00105005,
FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPT0_HASH = 0x00105006,
FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPT0_ENCRYPTION_VER = 0x00105007,
FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPT0_HASH_VER = 0x00105008,
FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPT0_PFS = 0x00105020,
FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPT0_PROTOCOL = 0x00105021,
FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPT0_ENCRYPTION = 0x00105022,
FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPT0_HASH = 0x00105023,

```

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPTO_TIMEOUT_MINUTES = 0x00105024,
FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPTO_TIMEOUT_KBYTES = 0x00105025,
FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPTO_ENCRYPTION_VER = 0x00105026,
FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPTO_HASH_VER = 0x00105027,
FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_OR_AND_CONDITIONS = 0x00106000,
FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_AND_CONDITIONS = 0x00106001,
FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_CONDITION_KEY = 0x00106002,
FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_CONDITION_MATCH_TYPE = 0x00106003,
FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_CONDITION_DATA_TYPE = 0x00106004,
FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_CONDITION_KEY_AND_DATA_TYPE = 0x00106005,
FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_KEYS_PROTOCOL_PORT = 0x00106006,
FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_KEY_PROFILE = 0x00106007,
FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_KEY_STATUS = 0x00106008,
FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_KEY_FILTERID = 0x00106009,
FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_KEY_APP_PATH = 0x00106010,
FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_KEY_PROTOCOL = 0x00106011,
FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_KEY_LOCAL_PORT = 0x00106012,
FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_KEY_REMOTE_PORT = 0x00106013,
FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_KEY_SVC_NAME = 0x00106015,
FW_RULE_STATUS_SEMANTIC_ERROR_REQUIRE_IN_CLEAR_OUT_ON_TRANSPORT = 0x00107000,
FW_RULE_STATUS_SEMANTIC_ERROR_TUNNEL_BYPASS_TUNNEL_IF_SECURE_ON_TRANSPORT = 0x00107001,
FW_RULE_STATUS_SEMANTIC_ERROR_AUTH_NOENCAP_ON_TUNNEL = 0x00107002,
FW_RULE_STATUS_SEMANTIC_ERROR_AUTH_NOENCAP_ON_PSK = 0x00107003,
FW_RULE_STATUS_SEMANTIC_ERROR_CRYPTO_ENCR_HASH = 0x00105040,
FW_RULE_STATUS_SEMANTIC_ERROR_CRYPTO_ENCR_HASH_COMPAT = 0x00105041,
FW_RULE_STATUS_SEMANTIC_ERROR_SCHEMA_VERSION = 0x00105050,
FW_RULE_STATUS_SEMANTIC_ERROR = 0x00100000,
FW_RULE_STATUS_RUNTIME_ERROR_PHASE1_AUTH_NOT_FOUND = 0x00200001,
FW_RULE_STATUS_RUNTIME_ERROR_PHASE2_AUTH_NOT_FOUND = 0x00200002,
FW_RULE_STATUS_RUNTIME_ERROR_PHASE2_CRYPTO_NOT_FOUND = 0x00200003,
FW_RULE_STATUS_RUNTIME_ERROR_AUTH_MCHN_SHKEY_MISMATCH = 0x00200004,
FW_RULE_STATUS_RUNTIME_ERROR_PHASE1_CRYPTO_NOT_FOUND = 0x00200005,
FW_RULE_STATUS_RUNTIME_ERROR_AUTH_NOENCAP_ON_TUNNEL = 0x00200006,
FW_RULE_STATUS_RUNTIME_ERROR_AUTH_NOENCAP_ON_PSK = 0x00200007,
FW_RULE_STATUS_RUNTIME_ERROR = 0x00200000,
FW_RULE_STATUS_ERROR = FW_RULE_STATUS_PARSING_ERROR | FW_RULE_STATUS_SEMANTIC_ERROR |
FW_RULE_STATUS_RUNTIME_ERROR,
FW_RULE_STATUS_ALL = 0xFFFF0000
} FW_RULE_STATUS;

```

FW_RULE_STATUS_OK: The rule was parsed successfully from the store, is correctly constructed, and has no issue.

FW_RULE_STATUS_PARTIALLY_IGNORED: The rule has fields that the service can successfully ignore. The ignored fields can be present only if the policy (such as the Group Policy) was written by future firewall and advanced security components that support a higher schema version. Therefore, this error occurs only if the version of the rule is higher; specifically, a higher minor version means that part of the rule might not be understandable. Because the host firewall component does not understand these new fields, it cannot meaningfully specify what was ignored in the rule.

FW_RULE_STATUS_IGNORED: The rule has a higher major version that the service MUST ignore. Higher major schema versions specify that nothing in the rule is understandable to lower major version components.

FW_RULE_STATUS_PARSING_ERROR_NAME: The name contains characters that are not valid or the length is not valid.

FW_RULE_STATUS_PARSING_ERROR_DESC: The description contains characters that are not valid or the length is not valid.

FW_RULE_STATUS_PARSING_ERROR_APP: The application contains characters that are not valid or the length is not valid.

FW_RULE_STATUS_PARSING_ERROR_SVC: The service contains characters that are not valid or the length is not valid.

FW_RULE_STATUS_PARSING_ERROR_RMA: The remote machine authentication contains characters that are not valid or the length is not valid.

FW_RULE_STATUS_PARSING_ERROR_RUA: The remote user authentication contains characters that are not valid or the length is not valid.

FW_RULE_STATUS_PARSING_ERROR_EMBD: The embedded context contains characters that are not valid or the length is not valid.

FW_RULE_STATUS_PARSING_ERROR_RULE_ID: The rule ID contains characters that are not valid or the length is not valid.

FW_RULE_STATUS_PARSING_ERROR_PHASE1_AUTH: The Phase1 authentication set ID contains characters that are not valid or the length is not valid.

FW_RULE_STATUS_PARSING_ERROR_PHASE2_CRYPTO: The Phase2 cryptographic set ID contains characters that are not valid or the length is not valid.

FW_RULE_STATUS_PARSING_ERROR_PHASE2_AUTH: The Phase2 authentication set ID contains characters that are not valid or the length is not valid.

FW_RULE_STATUS_PARSING_ERROR_RESOLVE_APP: The application name cannot be resolved.

FW_RULE_STATUS_PARSING_ERROR_PHASE1_CRYPTO: The Phase1 cryptographic set ID contains characters that are not valid or the length is not valid.

FW_RULE_STATUS_PARSING_ERROR: The rule did not parse correctly.

FW_RULE_STATUS_SEMANTIC_ERROR_RULE_ID: Semantic error: The rule ID is not specified.

FW_RULE_STATUS_SEMANTIC_ERROR_PORTS: Semantic error: Mismatch in the number of ports and port buffers.

FW_RULE_STATUS_SEMANTIC_ERROR_PORT_KEYW: Semantic error: The port keyword is not valid.

FW_RULE_STATUS_SEMANTIC_ERROR_PORT_RANGE: Semantic error: End != Begin or port = 0.

FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_V4_SUBNETS: Semantic error: Mismatch in the number of v4 subnets and subnet buffers.

FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_V6_SUBNETS: Semantic error: Mismatch in the number of v6 subnets and subnet buffers.

FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_V4_RANGES: Semantic error: Mismatch in the number of v4 ranges and range buffers.

FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_V6_RANGES: Semantic error: Mismatch in the number of v6 ranges and range buffers.

FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_RANGE: Semantic error: End < Begin.

FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_MASK: Semantic error: The mask specified on a v4 subnet is not valid.

FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_PREFIX: Semantic error: The prefix specified on a v6 subnet is not valid.

FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_KEYW: Semantic error: The specified keyword is not valid.

FW_RULE_STATUS_SEMANTIC_ERROR_LADDR_PROP: Semantic error: A property on local addresses does not belong to the LocalAddress.

FW_RULE_STATUS_SEMANTIC_ERROR_RADDR_PROP: Semantic error: A property on remote addresses does not belong to the RemoteAddress.

FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_V6: Semantic error: An unspecified or loopback IPv6 address was specified.

FW_RULE_STATUS_SEMANTIC_ERROR_LADDR_INTF: Semantic error: A local address cannot be used together with either an interface or an interface type.

FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_V4: Semantic error: An unspecified or loopback IPv4 address was specified.

FW_RULE_STATUS_SEMANTIC_ERROR_TUNNEL_ENDPOINT_ADDR: Semantic error: An endpoint "any" cannot be specified for a tunnel mode rule.

FW_RULE_STATUS_SEMANTIC_ERROR_DTE_VER: Semantic error: An incorrect schema version was specified for using dynamic tunnel endpoints.

FW_RULE_STATUS_SEMANTIC_ERROR_DTE_MISMATCH_ADDR: Semantic error: The v4 and v6 tunnel endpoints are neither local nor remote endpoints.

FW_RULE_STATUS_SEMANTIC_ERROR_PROFILE: Semantic error: The profile type is not valid.

FW_RULE_STATUS_SEMANTIC_ERROR_ICMP: Semantic error: Mismatch in the number of ICMPs and ICMP buffers.

FW_RULE_STATUS_SEMANTIC_ERROR_ICMP_CODE: Semantic error: The specified ICMP code is not valid.

FW_RULE_STATUS_SEMANTIC_ERROR_IF_ID: Semantic error: Mismatch in the number of interfaces and interface buffers.

FW_RULE_STATUS_SEMANTIC_ERROR_IF_TYPE: Semantic error: The specified interface type is not valid.

FW_RULE_STATUS_SEMANTIC_ERROR_ACTION: Semantic error: The specified action is not valid.

FW_RULE_STATUS_SEMANTIC_ERROR_ALLOW_BYPASS: Semantic error: An allow-bypass action is specified, but the rule does not meet allow-bypass criteria (such as, the direction is inbound, authenticate/encrypt flags are set, or remote machine authentication is set).

FW_RULE_STATUS_SEMANTIC_ERROR_DO_NOT_SECURE: Semantic error: A DO_NOT_SECURE action is specified together with authentication or cryptographic sets.

FW_RULE_STATUS_SEMANTIC_ERROR_ACTION_BLOCK_IS_ENCRYPTED_SECURE: Semantic error: A block action was specified together with a require security or a require encryption action.

FW_RULE_STATUS_SEMANTIC_ERROR_DIR: Semantic error: The specified direction is not valid.

FW_RULE_STATUS_SEMANTIC_ERROR_PROT: Semantic error: The specified protocol is not valid.

FW_RULE_STATUS_SEMANTIC_ERROR_PROT_PROP: Semantic error: The protocol and protocol-dependent fields do not match.

FW_RULE_STATUS_SEMANTIC_ERROR_DEFER_EDGE_PROP: Semantic error: A Dynamic edge flag (either defer to app or defer to user) is set without having an edge flag set.

FW_RULE_STATUS_SEMANTIC_ERROR_ALLOW_BYPASS_OUTBOUND: Semantic error: An outbound allow-bypass action is specified, but the rule does not meet allow-bypass criteria (authenticate/encrypt flags set).

FW_RULE_STATUS_SEMANTIC_ERROR_DEFER_USER_INVALID_RULE: The rule does not allow the defer user property to be set.

FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS: Semantic error: The specified flags are not valid.

FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_AUTO_AUTH: Semantic error: The autogenerate flag is set, but no authentication flags are set.

FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_AUTO_BLOCK: Semantic error: The autogenerate flag is set, but the action is block.

FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_AUTO_DYN_RPC: Semantic error: The autogenerate flag is set together with the dynamic RPC flag.

FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_AUTHENTICATE_ENCRYPT: Semantic error: The authenticate and authenticate-encrypt flags are both specified.

FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_AUTH_WITH_ENC_NEGOTIATE_VER: Semantic error: The schema version is not compliant with the Authenticate with Encryption flag.

FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_AUTH_WITH_ENC_NEGOTIATE: Semantic error: The Authenticate with Encryption Negotiate flag is specified but the basic Authenticate with Encryption flag is not set.

FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_ESP_NO_ENCAP_VER: Semantic error: The schema version is not compliant with the Authenticate with No Encapsulation flag.

FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_ESP_NO_ENCAP: Semantic error: The Authenticate with No Encapsulation flag is specified but the basic Authenticate flag is not set.

FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_TUNNEL_AUTH_MODES_VER: Semantic error: The schema version is not compliant with the tunnel authentication modes.

FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_TUNNEL_AUTH_MODES: Semantic error: The tunnel authentication modes are specified by a lower-version client.

FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_IP_TLS_VER: Semantic error: The schema version is not compliant with the IP_TLS flag.

FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_PORTRANGE_VER: Semantic error: The schema version is not compliant with port range support.

FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_ADDRS_TRAVERSE_DEFER_VER: Semantic error: The schema version is not compliant with the FW_RULE_FLAGS_ROUTEABLE_ADDRS_TRAVERSE_DEFER_APP flag. For more information, see [2.2.34](#).

FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_AUTH_WITH_ENC_NEGOTIATE_OUTBOUND: Semantic error: The Authenticate with Encryption Negotiate flag is set for the outbound rule.

FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_AUTHENTICATE_WITH_OUTBOUND_BY_PASS_VER: Semantic error: The Outbound Authenticated bypass is not supported on this version.

FW_RULE_STATUS_SEMANTIC_ERROR_REMOTE_AUTH_LIST: Semantic error: An authorized remote machine or user list is specified, but the authenticate/encryption flags were not set.

FW_RULE_STATUS_SEMANTIC_ERROR_REMOTE_USER_LIST: Semantic error: An authorized remote user list is specified on an outbound direction.

FW_RULE_STATUS_SEMANTIC_ERROR_PLATFORM: Semantic error: The number of valid operating system platforms and the list of valid operating system platforms do not match.

FW_RULE_STATUS_SEMANTIC_ERROR_PLATFORM_OP_VER: Semantic error: Schema version not compliant with the platform operator used.

FW_RULE_STATUS_SEMANTIC_ERROR_PLATFORM_OP: Semantic error: Invalid platform operator used.

FW_RULE_STATUS_SEMANTIC_ERROR_DTE_NOANY_ADDR: Semantic error: DTE is specified but all tunnel endpoints are specified.

FW_RULE_STATUS_SEMANTIC_TUNNEL_EXEMPT_WITH_GATEWAY: Semantic error: DTM tunnel exemption specified with tunnel endpoint (gateways) address.

FW_RULE_STATUS_SEMANTIC_TUNNEL_EXEMPT_VER: Semantic error: Schema version not compliant with tunnel mode exemptions.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_AUTH_SET_ID: Semantic error: Phase1 authentication set ID is not specified.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPTO_SET_ID: Semantic error: Phase2 cryptographic set ID is not specified.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPTO_SET_ID: Semantic error: Phase1 cryptographic set ID is not specified.

FW_RULE_STATUS_SEMANTIC_ERROR_SET_ID: Semantic error: The set ID is not specified.

FW_RULE_STATUS_SEMANTIC_ERROR_IPSEC_PHASE: Semantic error: The specified phase is not valid.

FW_RULE_STATUS_SEMANTIC_ERROR_EMPTY_SUITES: Semantic error: No suites are specified in the set.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_AUTH_METHOD: Semantic error: The Phase1 authentication method is not valid.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_AUTH_METHOD: Semantic error: The Phase2 authentication method is not valid.

FW_RULE_STATUS_SEMANTIC_ERROR_AUTH_METHOD_ANONYMOUS: Semantic error: Anonymous authentication is specified as the only authentication proposal (or authentication proposal suite).

FW_RULE_STATUS_SEMANTIC_ERROR_AUTH_METHOD_DUPLICATE: Semantic error: Duplicate authentication methods are specified but not supported.

FW_RULE_STATUS_SEMANTIC_ERROR_AUTH_SUITE_FLAGS: Semantic error: The specified authentication suite flags are not valid.

FW_RULE_STATUS_SEMANTIC_ERROR_HEALTH_CERT: Semantic error: The machine certificate MUST be a health certificate for Phase2 authentication.

FW_RULE_STATUS_SEMANTIC_ERROR_AUTH_SIGNCERT_VER: Semantic error: The suite specifies signing that is not compliant with its schema version.

FW_RULE_STATUS_SEMANTIC_ERROR_AUTH_INTERMEDIATE_CA_VER: Semantic error: Specifies an intermediate **certificate authority (CA)** that is not compliant with its schema version.

FW_RULE_STATUS_SEMANTIC_ERROR_MACHINE_SHKEY: Semantic error: The machine shared key is either missing or not valid.

FW_RULE_STATUS_SEMANTIC_ERROR_CA_NAME: Semantic error: The CA name is either missing or not valid.

FW_RULE_STATUS_SEMANTIC_ERROR_MIXED_CERTS: Semantic error: Health certificates (CERTS) cannot be specified together with regular certificates.

FW_RULE_STATUS_SEMANTIC_ERROR_NON_CONTIGUOUS_CERTS: Semantic error: Certificates that have a specific signing algorithm are not contiguous.

FW_RULE_STATUS_SEMANTIC_ERROR_MIXED_CA_TYPE_IN_BLOCK: Semantic error: Both root and intermediate CA types cannot be present in the same signing algorithm block.

FW_RULE_STATUS_SEMANTIC_ERROR_MACHINE_USER_AUTH: Semantic error: Both machine and user authentications are specified.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPTONONDEFAULT_ID: Semantic error: The ID for the Phase1 cryptographic set is not the default.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPTOFLAGS: Semantic error: The Phase1 cryptographic set flags are not valid.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPTO_TIMEOUT_MINUTES: Semantic error: The Phase1 cryptographic set time-out minutes are not valid.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPTO_TIMEOUT_SESSIONS: Semantic error: The time-out sessions for the Phase1 cryptographic set are not valid.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPTO_KEY_EXCHANGE: Semantic error: The key exchange for the Phase1 cryptographic set is not valid.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPTO_ENCRYPTION: Semantic error: The Phase1 cryptographic set encryption is not valid.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPTO_HASH: Semantic error: The Phase1 cryptographic set hash is not valid.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPTO_ENCRYPTION_VER: Semantic error: The Phase1 cryptographic set encryption is not schema-version compliant.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPTO_HASH_VER: Semantic error: The Phase1 cryptographic set hash is not schema version compliant.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPTO_PFS: Semantic error: The Phase2 cryptographic set **perfect forward secrecy (PFS)** is not valid.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPTO_PROTOCOL: Semantic error: The Phase2 cryptographic set protocol is not valid.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPTO_ENCRYPTION: Semantic error: The Phase2 cryptographic set encryption is not valid.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPTO_HASH: Semantic error: The Phase2 cryptographic set hash is not valid.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPTO_TIMEOUT_MINUTES: Semantic error: The Phase2 cryptographic set time-out minutes are not valid.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPTO_TIMEOUT_KBYTES: Semantic error: The Phase2 cryptographic set time-out kilobytes are not valid.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPTO_ENCRYPTION_VER: Semantic error: The Phase2 cryptographic set encryption is not schema-version compliant.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPTO_HASH_VER: The Phase2 cryptographic set hash is not schema-version compliant.

FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_OR_AND_CONDITIONS: Semantic error: A mismatch exists in the number of OR'd terms and term arrays.

FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_AND_CONDITIONS: Semantic error: A mismatch exists in the number of AND'd conditions and condition arrays.

FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_CONDITION_KEY: Semantic error: The condition match key is not valid.

FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_CONDITION_MATCH_TYPE: Semantic error: The condition match type is not valid.

FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_CONDITION_DATA_TYPE: Semantic error: The condition data type is not valid.

FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_CONDITION_KEY_AND_DATA_TYPE: Semantic error: The key and data type combination is not valid.

FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_KEYS_PROTOCOL_PORT: Semantic error: A port condition is present without a protocol condition.

FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_KEY_PROFILE: Semantic error: The profile key is unavailable for the queried object type.

FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_KEY_STATUS: Semantic error: The status key is unavailable for the queried object type.

FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_KEY_FILTERID: Semantic error: The FilterID key is unavailable for the queried object type.

FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_KEY_APP_PATH: Semantic error: The application key is unavailable for the queried object type.

FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_KEY_PROTOCOL: Semantic error: The protocol key is unavailable for the queried object type.

FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_KEY_LOCAL_PORT: Semantic error: The local port key is unavailable for the queried object type.

FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_KEY_REMOTE_PORT: Semantic error: The remote port key is unavailable for the queried object type.

FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_KEY_SVC_NAME: Semantic error: The service name key is unavailable for the queried object type.

FW_RULE_STATUS_SEMANTIC_ERROR_REQUIRE_IN_CLEAR_OUT_ON_TRANSPORT: Semantic error: "Require in clear out" tunnel authentication mode cannot be set on transport mode rules.

FW_RULE_STATUS_SEMANTIC_ERROR_TUNNEL_BYPASS_TUNNEL_IF_SECURE_ON_TRANSPORT: Semantic error: Cannot set flag to exempt IPsec transport traffic from a tunnel mode, on a transport rule.

FW_RULE_STATUS_SEMANTIC_ERROR_AUTH_NOENCAP_ON_TUNNEL: Semantic error: Cannot set FW_CRYPTOPROTOCOL_AUTH_NO_ENCAP (see section [2.2.61](#)) on a tunnel mode rule.

FW_RULE_STATUS_SEMANTIC_ERROR_AUTH_NOENCAP_ON_PSK: Semantic error: Cannot mix FW_CRYPTOPROTOCOL_AUTH_NO_ENCAP (see section [2.2.61](#)) protocol with Preshared key authentication methods.

FW_RULE_STATUS_SEMANTIC_ERROR_CRYPTO_ENCR_HASH: Semantic error: Both the encryption and hash are not specified.

FW_RULE_STATUS_SEMANTIC_ERROR_CRYPTO_ENCR_HASH_COMPAT: Semantic error: The encryption and hash use incompatible algorithms.

FW_RULE_STATUS_SEMANTIC_ERROR_SCHEMA_VERSION: Semantic error: The specified schema version is earlier than the supported versions.

FW_RULE_STATUS_SEMANTIC_ERROR: There is a semantic error when considering the fields of the rule in conjunction with other policy objects.

FW_RULE_STATUS_RUNTIME_ERROR_PHASE1_AUTH_NOT_FOUND: A Phase1 authentication set is not found.

FW_RULE_STATUS_RUNTIME_ERROR_PHASE2_AUTH_NOT_FOUND: A Phase2 authentication set is not found.

FW_RULE_STATUS_RUNTIME_ERROR_PHASE2_CRYPTO_NOT_FOUND: A Phase2 cryptographic set is not found.

FW_RULE_STATUS_RUNTIME_ERROR_AUTH_MCHN_SHKEY_MISMATCH: A Phase2 authentication set cannot be specified when the Phase1 authentication set contains a pre-shared key as an authentication method.

FW_RULE_STATUS_RUNTIME_ERROR_PHASE1_CRYPTO_NOT_FOUND: A Phase1 cryptographic set is not found.

FW_RULE_STATUS_RUNTIME_ERROR_AUTH_NOENCAP_ON_TUNNEL: Semantic error: Cannot set FW_CRYPTOPROTOCOL_AUTH_NO_ENCAP (see section [2.2.61](#)) on a tunnel mode rule.

FW_RULE_STATUS_RUNTIME_ERROR_AUTH_NOENCAP_ON_PSK: Semantic error: Cannot mix FW_CRYPTOPROTOCOL_AUTH_NO_ENCAP (see section [2.2.61](#)) protocol with Preshared key authentication methods.

FW_RULE_STATUS_RUNTIME_ERROR: There is a runtime error when the object is considered with other policy objects.

FW_RULE_STATUS_ERROR: An error of any kind occurred. This symbolic constant has a value of 0x00380000.

FW_RULE_STATUS_ALL: The status of all (it is used to enumerate all the rules, regardless of the status).

2.2.24 FW_RULE_STATUS_CLASS

This enumeration defines classes of status codes.

```
typedef enum _tag_FW_RULE_STATUS_CLASS
{
    FW_RULE_STATUS_CLASS_OK = FW_RULE_STATUS_OK,
    FW_RULE_STATUS_CLASS_PARTIALLY_IGNORED = FW_RULE_STATUS_PARTIALLY_IGNORED,
    FW_RULE_STATUS_CLASS_IGNORED = FW_RULE_STATUS_IGNORED,
    FW_RULE_STATUS_CLASS_PARSING_ERROR = FW_RULE_STATUS_PARSING_ERROR,
    FW_RULE_STATUS_CLASS_SEMANTIC_ERROR = FW_RULE_STATUS_SEMANTIC_ERROR,
    FW_RULE_STATUS_CLASS_RUNTIME_ERROR = FW_RULE_STATUS_RUNTIME_ERROR,
    FW_RULE_STATUS_CLASS_ERROR = FW_RULE_STATUS_ERROR,
    FW_RULE_STATUS_CLASS_ALL = FW_RULE_STATUS_ALL
} FW_RULE_STATUS_CLASS;
```

FW_RULE_STATUS_CLASS_OK: The rule is correctly constructed and has no issue. This symbolic constant has a value of 0x00010000.

FW_RULE_STATUS_CLASS_PARTIALLY_IGNORED: The rule has fields that the service can successfully ignore. This symbolic constant has a value of 0x00020000.

FW_RULE_STATUS_CLASS_IGNORED: The rule has a higher version that the service MUST ignore. This symbolic constant has a value of 0x00040000.

FW_RULE_STATUS_CLASS_PARSING_ERROR: The rule failed to be parsed correctly. This symbolic constant has a value of 0x00080000.

FW_RULE_STATUS_CLASS_SEMANTIC_ERROR: There is a semantic error when considering the fields of the rule in conjunction. This symbolic constant has a value of 0x00100000.

FW_RULE_STATUS_CLASS_RUNTIME_ERROR: There is a runtime error when the object is considered in conjunction with other policy objects. This symbolic constant has a value of 0x00200000.

FW_RULE_STATUS_CLASS_ERROR: An error occurred. This symbolic constant has a value of 0x00380000.

FW_RULE_STATUS_CLASS_ALL: The status of all (used to enumerate ALL the rules, regardless of the status). This symbolic constant has a value of 0xFFFF0000.

2.2.25 FW_OBJECT_CTRL_FLAG

This enumeration is used to indicate the RPC protocol when elements in structures are included.

```
typedef enum _tag_FW_OBJECT_CTRL_FLAG
{
    FW_OBJECT_CTRL_FLAG_INCLUDE_METADATA = 0x0001
} FW_OBJECT_CTRL_FLAG;
```

FW_OBJECT_CTRL_FLAG_INCLUDE_METADATA: This flag indicates that the structure where this flag is specified contains metadata information.

2.2.26 FW_ENFORCEMENT_STATE

This enumeration is part of the metadata information. It provides information about how an object is using enforcement state objects, the current status of the object, and a reason for a state at a specific point in time.

```
typedef enum _tag_FW_ENFORCEMENT_STA
{
    FW_ENFORCEMENT_STATE_INVALID,
    FW_ENFORCEMENT_STATE_FULL,
    FW_ENFORCEMENT_STATE_WF_OFF_IN_PROFILE,
    FW_ENFORCEMENT_STATE_CATEGORY_OFF,
    FW_ENFORCEMENT_STATE_DISABLED_OBJECT,
    FW_ENFORCEMENT_STATE_INACTIVE_PROFILE,
    FW_ENFORCEMENT_STATE_LOCAL_ADDRESS_RESOLUTION_EMPTY,
    FW_ENFORCEMENT_STATE_REMOTE_ADDRESS_RESOLUTION_EMPTY,
    FW_ENFORCEMENT_STATE_LOCAL_PORT_RESOLUTION_EMPTY,
    FW_ENFORCEMENT_STATE_REMOTE_PORT_RESOLUTION_EMPTY,
    FW_ENFORCEMENT_STATE_INTERFACE_RESOLUTION_EMPTY,
    FW_ENFORCEMENT_STATE_APPLICATION_RESOLUTION_EMPTY,
    FW_ENFORCEMENT_STATE_REMOTE_MACHINE_EMPTY,
    FW_ENFORCEMENT_STATE_REMOTE_USER_EMPTY,
```

```
FW_ENFORCEMENT_STATE_LOCAL_GLOBAL_OPEN_PORTS_DISALLOWED,  
FW_ENFORCEMENT_STATE_LOCAL_AUTHORIZED_APPLICATIONS_DISALLOWED,  
FW_ENFORCEMENT_STATE_LOCAL_FIREWALL_RULES_DISALLOWED,  
FW_ENFORCEMENT_STATE_LOCAL_CONSEC_RULES_DISALLOWED,  
FW_ENFORCEMENT_STATE_MISMATCHED_PLATFORM,  
FW_ENFORCEMENT_STATE_OPTIMIZED_OUT,  
FW_ENFORCEMENT_STATE_MAX  
} FW_ENFORCEMENT_STATE;
```

FW_ENFORCEMENT_STATE_INVALID: This value is invalid and MUST NOT be used by the server. It is defined for simplicity in writing IDL definitions and code. This symbolic constant has a value of 0.

FW_ENFORCEMENT_STATE_FULL: The object fully generated all enforcement state objects to implement its full functionality. This symbolic constant has a value of 1.

FW_ENFORCEMENT_STATE_WF_OFF_IN_PROFILE: The object did not generate some enforcement state objects because the firewall and advanced security component is not active in a profile where the object is meant to be applied. This symbolic constant has a value of 2.

FW_ENFORCEMENT_STATE_CATEGORY_OFF: The object did not generate some enforcement state objects because a third-party software component registered with the firewall and advanced security component to own the functionality that the object is meant to perform. This symbolic constant has a value of 3.

FW_ENFORCEMENT_STATE_DISABLED_OBJECT: The object did not generate some enforcement state objects because the object is disabled. This symbolic constant has a value of 4.

FW_ENFORCEMENT_STATE_INACTIVE_PROFILE: The object did not generate some enforcement state objects because one of the profiles that the object is meant to be applied to is not currently active. This symbolic constant has a value of 5.

FW_ENFORCEMENT_STATE_LOCAL_ADDRESS_RESOLUTION_EMPTY: The object did not generate some enforcement state objects because the local address condition of the object contains a keyword that resolves to an empty set. This symbolic constant has a value of 6.

FW_ENFORCEMENT_STATE_REMOTE_ADDRESS_RESOLUTION_EMPTY: The object did not generate some enforcement state objects because the remote address condition of the object contains a keyword that resolves to an empty set. This symbolic constant has a value of 7.

FW_ENFORCEMENT_STATE_LOCAL_PORT_RESOLUTION_EMPTY: The object did not generate some enforcement state objects because the local port condition of the object contains a keyword that resolves to an empty set. This symbolic constant has a value of 8.

FW_ENFORCEMENT_STATE_REMOTE_PORT_RESOLUTION_EMPTY: The object did not generate some enforcement state objects because the remote port condition of the object contains a keyword that resolves to an empty set. This symbolic constant has a value of 9.

FW_ENFORCEMENT_STATE_INTERFACE_RESOLUTION_EMPTY: The object did not generate some enforcement state objects because the interface condition of the object contains a keyword that resolves to an empty set. This symbolic constant has a value of 10.

FW_ENFORCEMENT_STATE_APPLICATION_RESOLUTION_EMPTY: The object did not generate some enforcement state objects because the application condition of the object

contains a path that could not resolve to a valid file system path. This symbolic constant has a value of 11.

FW_ENFORCEMENT_STATE_REMOTE_MACHINE_EMPTY: The object did not generate some enforcement state objects because the remote machine condition of the object contains an SDDL with a **security identifier (SID)** that is not currently available on the host. This symbolic constant has a value of 12.

FW_ENFORCEMENT_STATE_REMOTE_USER_EMPTY: The object did not generate some enforcement state objects because the remote user condition of the object contains an SDDL with a (SID) that is not currently available on the host. This symbolic constant has a value of 13.

FW_ENFORCEMENT_STATE_LOCAL_GLOBAL_OPEN_PORTS_DISALLOWED: The object did not generate some enforcement state objects because the FW_PROFILE_CONFIG_AUTH_APPS_ALLOW_USER_PREF_MERGE configuration option (see section [2.2.37](#) for more details) from a profile that the object applied to, disallowed its use. This symbolic constant has a value of 14.

FW_ENFORCEMENT_STATE_LOCAL_AUTHORIZED_APPLICATIONS_DISALLOWED: The object did not generate some enforcement state objects because the FW_PROFILE_CONFIG_GLOBAL_PORTS_ALLOW_USER_PREF_MERGE configuration option (see section [2.2.37](#) for more details) from a profile that the object applied to, disallowed its use. This symbolic constant has a value of 15.

FW_ENFORCEMENT_STATE_LOCAL_FIREWALL_RULES_DISALLOWED: The object did not generate some enforcement state objects because the FW_PROFILE_CONFIG_ALLOW_LOCAL_POLICY_MERGE configuration option (see section [2.2.37](#) for more details) from a profile that the object applied to, disallowed its use. This symbolic constant has a value of 16.

FW_ENFORCEMENT_STATE_LOCAL_CONSEC_RULES_DISALLOWED: The object did not generate some enforcement state objects because the FW_PROFILE_CONFIG_ALLOW_LOCAL_IPSEC_POLICY_MERGE configuration option (see section [2.2.37](#) for more details) from a profile that the object applied to, disallowed its use. This symbolic constant has a value of 17.

FW_ENFORCEMENT_STATE_MISMATCHED_PLATFORM: The object did not generate some enforcement state objects because the platform validity condition does not match the current platform of the host. This symbolic constant has a value of 18.

FW_ENFORCEMENT_STATE_OPTIMIZED_OUT: The object did not generate some enforcement state objects because the firewall and advanced security component determined that the object-implemented functionality is irrelevant (would not change or affect what traffic is allowed or permitted) at the current time. Therefore, the component optimized out the irrelevant functionality and ignored it. This is a pure optimization. This symbolic constant has a value of 19.

FW_ENFORCEMENT_STATE_MAX: This value and greater values are invalid and MUST NOT be used. This symbolic constant is defined for simplicity in writing IDL definitions and code. It has a value of 20.

2.2.27 FW_OBJECT_METADATA

This structure contains the metadata that is associated with a specific policy object.

```

typedef struct _tag_FW_OBJECT_METADATA {
    unsigned __int64 qwFilterContextID;
    [range(0, 100)] DWORD dwNumEntries;
    [size_is(dwNumEntries)] FW_ENFORCEMENT_STATE* pEnforcementStates;
} FW_OBJECT_METADATA,
*PFW_OBJECT_METADATA;

```

qwFilterContextID: This field is not used across the wires.

dwNumEntries: A field that specifies the number of metadata hints ([FW_ENFORCEMENT_STATES](#)) that the structure contains.

pEnforcementStates: A pointer to an array of **FW_ENFORCEMENT_STATE** elements. The number of elements is given by **dwNumEntries**.

2.2.28 FW_OS_PLATFORM_OP

This enumeration describes the operations used in the [FW_OS_PLATFORM](#) structure to determine if an object should be applied to a specified operating system platform.

```

typedef enum
{
    FW_OS_PLATFORM_OP_EQ,
    FW_OS_PLATFORM_OP_GTEQ,
    FW_OS_PLATFORM_OP_MAX
} FW_OS_PLATFORM_OP;

```

FW_OS_PLATFORM_OP_EQ: The operating system platform MUST be the same as the one specified. This is satisfied when the following occurs:

- If (((bPlatform & 0x7) == platform type) && (bMajorVersion == major version) && (bMinorVersion == minor version)).

Otherwise, the operating system is not equal to the one specified. This symbolic constant has a value of 0.

FW_OS_PLATFORM_OP_GTEQ: The operating system MUST be greater than or equal to the one specified. This is satisfied when any of the following occur:

- If (bPlatform & 0x7) > platform type
- If (((bPlatform & 0x7) == platform type) && (bMajorVersion > major version))
- If (((bPlatform & 0x7) == platform type) && (bMajorVersion == major version) && (bMinorVersion >= minor version))

Otherwise, the operation system is not greater than or equal to the one specified. This symbolic constant has a value of 1.

FW_OS_PLATFORM_OP_MAX: This value and greater values are invalid and MUST NOT be used. This symbolic constant is defined for simplicity in writing IDL definitions and code. It has a value of 2.

2.2.29 FW_OS_PLATFORM

This structure identifies an operating system platform or platforms. <3>

```
typedef struct _tag_FW_OS_PLATFORM {
    unsigned char bPlatform;
    unsigned char bMajorVersion;
    unsigned char bMinorVersion;
    unsigned char Reserved;
} FW_OS_PLATFORM,
*PFW_OS_PLATFORM;
```

bPlatform: The three least significant bits identify the platform type. The five most significant bits contain a value from the [FW_OS_PLATFORM_OP](#) enumeration.

bMajorVersion: Specifies the major version number for the OS.

bMinorVersion: Specifies the minor version number for the OS.

Reserved: Not used. Reserved for future use.

2.2.30 FW_OS_PLATFORM_LIST

This structure is used to contain a number of [FW_OS_PLATFORM](#) elements.

```
typedef struct _tag_FW_OS_PLATFORM_LIST {
    [range(0, 1000)] unsigned long dwNumEntries;
    [size_is(dwNumEntries)] PFW_OS_PLATFORM pPlatforms;
} FW_OS_PLATFORM_LIST,
*PFW_OS_PLATFORM_LIST;
```

dwNumEntries: This field specifies the number of OS platforms that the structure contains.

pPlatforms: A pointer to an array of **dwNumEntries** contiguous **FW_OS_PLATFORM** elements.

2.2.31 FW_RULE_ORIGIN_TYPE

This enumeration represents where the policy object is stored and from where it originates.

```
typedef enum _tag_FW_RULE_ORIGIN_TYPE
{
    FW_RULE_ORIGIN_INVALID,
    FW_RULE_ORIGIN_LOCAL,
    FW_RULE_ORIGIN_GP,
    FW_RULE_ORIGIN_DYNAMIC,
    FW_RULE_ORIGIN_AUTOGEN,
    FW_RULE_ORIGIN_HARDCODED,
    FW_RULE_ORIGIN_MAX
} FW_RULE_ORIGIN_TYPE;
```


FW_RULE_ORIGIN_INVALID: On enumeration, this value is invalid, and MUST NOT be used by the server. It is defined for simplicity in writing IDL definitions and code. However, the server ignores the fields of this data type on input, and hence it is valid for filling rules. This symbolic constant has a value of 0.

FW_RULE_ORIGIN_LOCAL: Specifies that the policy object originates from the local store. This symbolic constant has a value of 1.

FW_RULE_ORIGIN_GP: Specifies that the policy object originates from the GP store. This symbolic constant has a value of 2.

FW_RULE_ORIGIN_DYNAMIC: Specifies that the policy object originates from the dynamic store. This symbolic constant has a value of 3.

FW_RULE_ORIGIN_AUTOGEN: Not used. This symbolic constant has a value of 4.

FW_RULE_ORIGIN_HARDCODED: Specifies that the policy object originates from the firewall and advanced security component hard-coded values and is used due to lack of user settings. These values are not configurable and are not addressed in this protocol specification. Specific implementations of firewall and advanced security components can choose what hard-coded values to use when no other user settings are available. The only policy objects in this protocol specification that can have this FW_RULE_ORIGIN_HARDCODED value assigned are authentication sets and cryptographic sets, which are defined in sections [2.2.57](#) and [2.2.66](#), respectively. [<4>](#) This symbolic constant has a value of 5.

FW_RULE_ORIGIN_MAX: This value and greater values are invalid and MUST NOT be used. This symbolic constant is defined for simplicity in writing IDL definitions and code. It has a value of 6.

2.2.32 FW_ENUM_RULES_FLAGS

This enumeration defines flag values that can be used in the enumeration methods that are defined in [RRPC FWEnumFirewallRules](#), [RRPC FWEnumConnectionSecurityRules](#), [RRPC FWEnumAuthenticationSets](#), and [RRPC FWEnumCryptoSets](#).

```
typedef enum _tag_FW_ENUM_RULES_FLAGS
{
    FW_ENUM_RULES_FLAG_NONE = 0x0000,
    FW_ENUM_RULES_FLAG_RESOLVE_NAME = 0x0001,
    FW_ENUM_RULES_FLAG_RESOLVE_DESCRIPTION = 0x0002,
    FW_ENUM_RULES_FLAG_RESOLVE_APPLICATION = 0x0004,
    FW_ENUM_RULES_FLAG_RESOLVE_KEYWORD = 0x0008,
    FW_ENUM_RULES_FLAG_RESOLVE_GPO_NAME = 0x0010,
    FW_ENUM_RULES_FLAG_EFFECTIVE = 0x0020,
    FW_ENUM_RULES_FLAG_INCLUDE_METADATA = 0x0040,
    FW_ENUM_RULES_FLAG_MAX = 0x0080
} FW_ENUM_RULES_FLAGS;
```

FW_ENUM_RULES_FLAG_NONE: This value signifies that no specific flag is used. It is defined for IDL definitions and code to add readability, instead of using the number 0.

FW_ENUM_RULES_FLAG_RESOLVE_NAME: Resolves rule name strings to user-friendly, localizable strings if they are in the following format: @file.dll,-<resID>.

FW_ENUM_RULES_FLAG_RESOLVE_DESCRIPTION: Resolves rule description strings to user-friendly, localizable strings if they are in the following format: @file.dll,-<resID>.

FW_ENUM_RULES_FLAG_RESOLVE_APPLICATION: Resolves environment variables in the application string.

FW_ENUM_RULES_FLAG_RESOLVE_KEYWORD: Resolves keywords in addresses and ports to the actual addresses and ports (dynamic store only).

FW_ENUM_RULES_FLAG_RESOLVE_GPO_NAME: Resolves the GPO name for the GP_RSOP rules.

FW_ENUM_RULES_FLAG_EFFECTIVE: Enumerates rules only if an attempt was made to generate enforcement state objects<5> (defined in section [3.1.1](#)). This flag is available for the dynamic store only.

FW_ENUM_RULES_FLAG_INCLUDE_METADATA: Includes the metadata object information, represented by the **FW_OBJECT_METADATA** structure, in the enumerated objects.

FW_ENUM_RULES_FLAG_MAX: This value and greater values are invalid and MUST NOT be used. It is defined for simplicity in writing IDL definitions and code.

2.2.33 FW_RULE_ACTION

This enumeration describes the possible actions on firewall rules.

```
typedef enum _tag_FW_RULE_ACTION
{
    FW_RULE_ACTION_INVALID = 0,
    FW_RULE_ACTION_ALLOW_BYPASS,
    FW_RULE_ACTION_BLOCK,
    FW_RULE_ACTION_ALLOW,
    FW_RULE_ACTION_MAX
} FW_RULE_ACTION;
```

FW_RULE_ACTION_INVALID: This value is invalid and MUST NOT be used. It is defined for simplicity in writing IDL definitions and code. This symbolic constant has a value of zero.

FW_RULE_ACTION_ALLOW_BYPASS: Rules with this action allow traffic but are applicable only to rules that at least specify the **FW_RULE_FLAGS_AUTHENTICATE** flag. This symbolic constant has a value of 1.

FW_RULE_ACTION_BLOCK: Rules with this action block traffic. This symbolic constant has a value of 2.

FW_RULE_ACTION_ALLOW: Rules with this action allow traffic. This symbolic constant has a value of 3.

FW_RULE_ACTION_MAX: This value and greater values are invalid and MUST NOT be used. This symbolic constant is defined for simplicity in writing IDL definitions and code. It has a value of 4.

The actions also specify order of evaluation within the firewall rules. Allow-bypass rules are evaluated first, block rules second, and allow rules after that. (The last action to evaluate is the default action.)

2.2.34 FW_RULE_FLAGS

This enumeration represents flags that can be specified in firewall rules of section [2.2.36](#).

```

typedef enum _tag_FW_RULE_FLAGS
{
    FW_RULE_FLAGS_NONE = 0x0000,
    FW_RULE_FLAGS_ACTIVE = 0x0001,
    FW_RULE_FLAGS_AUTHENTICATE = 0x0002,
    FW_RULE_FLAGS_AUTHENTICATE_WITH_ENCRYPTION = 0x0004,
    FW_RULE_FLAGS_ROUTEABLE_ADDRS_TRAVERSE = 0x0008,
    FW_RULE_FLAGS_LOOSE_SOURCE_MAPPED = 0x0010,
    FW_RULE_FLAGS_AUTH_WITH_NO_ENCAPSULATION = 0x0020,
    FW_RULE_FLAGS_AUTH_WITH_ENC_NEGOTIATE = 0x0040,
    FW_RULE_FLAGS_ROUTEABLE_ADDRS_TRAVERSE_DEFER_APP = 0x0080,
    FW_RULE_FLAGS_ROUTEABLE_ADDRS_TRAVERSE_DEFER_USER = 0x0100,
    FW_RULE_FLAGS_AUTHENTICATE_BYPASS_OUTBOUND = 0x0200,
    FW_RULE_FLAGS_MAX = 0x0400,
    FW_RULE_FLAGS_MAX_V2_1 = 0x0020,
    FW_RULE_FLAGS_MAX_V2_9 = 0x0040
} FW_RULE_FLAGS;

```

FW_RULE_FLAGS_NONE: This value means that none of the following flags are set. It is defined for simplicity in writing IDL definitions and code.

FW_RULE_FLAGS_ACTIVE: The rule is enabled if this flag is set; otherwise, it is disabled.

FW_RULE_FLAGS_AUTHENTICATE: This flag MUST be set only on rules that have the allow actions. If set, traffic that matches the rule is allowed only if it has been authenticated by IPsec; otherwise, traffic is blocked.

FW_RULE_FLAGS_AUTHENTICATE_WITH_ENCRYPTION: This flag is similar to the FW_RULE_FLAGS_AUTHENTICATE flag; however, traffic MUST also be encrypted.

FW_RULE_FLAGS_ROUTEABLE_ADDRS_TRAVERSE: This flag MUST be set only on inbound rules. This flag allows the matching traffic to traverse a NAT edge device and be allowed in the host computer.

FW_RULE_FLAGS_LOOSE_SOURCE_MAPPED: This flag allows responses from a remote IP address that is different from the one to which the outbound matched traffic originally went.

FW_RULE_FLAGS_AUTH_WITH_NO_ENCAPSULATION: This flag MUST be set only on rules that have the FW_RULE_FLAGS_AUTHENTICATE flag set. If set, traffic that matches the rule is allowed if IKE or AuthIP authentication was successful; however, this flag does not necessarily require that traffic be protected by IPsec encapsulations.

FW_RULE_FLAGS_AUTH_WITH_ENC_NEGOTIATE: This flag MUST be set only on inbound rules that have the FW_RULE_FLAGS_AUTHENTICATE_WITH_ENCRYPTION flag set. If set and if the first packet that arrives is unencrypted but authenticated by IPsec, the packet is allowed, and an IKE or AuthIP negotiation is started to negotiate encryption settings and encrypt subsequent packets. [\[MS-AIPS\]](#) section 3.2.4 specifies negotiation initiation behavior for hosts that support both IKE and AuthIP negotiation. If the negotiation fails, the connection is dropped.

FW_RULE_FLAGS_ROUTEABLE_ADDRS_TRAVERSE_DEFER_APP: This flag MUST be set only on inbound rules. This flag allows the matching traffic to traverse a NAT edge device and be allowed in the host computer, if and only if, the software component listening on this traffic specifically requested NAT traversal traffic. The operating system implements how this request is made. [<6>](#)

FW_RULE_FLAGS_ROUTEABLE_ADDRS_TRAVERSE_DEFER_USER: This flag MUST be set only on inbound rules. Whenever an application tries to listen for traffic that matches this rule, the operating system asks the administrator of the host whether it should allow this traffic to traverse the NAT.

FW_RULE_FLAGS_AUTHENTICATE_BYPASS_OUTBOUND: This flag MUST be set only on outbound rules that have an allow action with either the FW_RULE_FLAGS_AUTHENTICATE or the FW_RULE_FLAGS_AUTHENTICATE_WITH_ENCRYPTION flag set. If set, this rule is evaluated before block rules, making it equivalent to a rule with an FW_RULE_ACTION_ALLOW_BYPASS, but for outbound.

FW_RULE_FLAGS_MAX: This value and greater values are invalid and MUST NOT be used. This value is defined for simplicity in writing IDL definitions and code.

FW_RULE_FLAGS_MAX_V2_1: This value and greater values are invalid and MUST NOT be used. This value is defined for simplicity in writing IDL definitions and code.

FW_RULE_FLAGS_MAX_V2_9: This value and greater values are invalid and MUST NOT be used. This value is defined for simplicity in writing IDL definitions and code.

2.2.35 FW_RULE2_0

This structure represents a firewall rule that is used by the 2.0 binary version servers and clients (see sections [1.7](#) and [2.2.41](#)). The fields of this structure are identical to the [FW_RULE](#) structure and its meanings are covered in section [2.2.36](#).

```
typedef struct _tag_FW_RULE2_0 {
    struct _tag_FW_RULE2_0* pNext;
    unsigned short wsSchemaVersion;
    [string, range(1, 10001), ref] wchar_t* wszRuleId;
    [string, range(1, 10001)] wchar_t* wszName;
    [string, range(1, 10001)] wchar_t* wszDescription;
    unsigned long dwProfiles;
    [range(FW_DIR_INVALID, FW_DIR_OUT)]
        FW_DIRECTION Direction;
    [range(0, 256)] unsigned short wIpProtocol;
    [switch_type(unsigned short), switch_is(wIpProtocol)]
        union {
            [case(6,17)]
                struct {
                    FW_PORTS LocalPorts;
                    FW_PORTS RemotePorts;
                };
            [case(1)]
                FW_ICMP_TYPE_CODE_LIST V4TypeCodeList;
            [case(58)]
                FW_ICMP_TYPE_CODE_LIST V6TypeCodeList;
            [default]
                ;
        };
    FW_ADDRESSES LocalAddresses;
    FW_ADDRESSES RemoteAddresses;
    FW_INTERFACE_LUIDS LocalInterfaceIds;
    unsigned long dwLocalInterfaceTypes;
    [string, range(1, 10001)] wchar_t* wszLocalApplication;
    [string, range(1, 10001)] wchar_t* wszLocalService;
    [range(FW_RULE_ACTION_INVALID, FW_RULE_ACTION_MAX)]
        FW_RULE_ACTION Action;
};
```

```

unsigned short wFlags;
[string, range(1, 10001)] wchar_t* wszRemoteMachineAuthorizationList;
[string, range(1, 10001)] wchar_t* wszRemoteUserAuthorizationList;
[string, range(1, 10001)] wchar_t* wszEmbeddedContext;
FW_OS_PLATFORM_LIST PlatformValidityList;
FW_RULE_STATUS Status;
[range(FW_RULE_ORIGIN_INVALID, FW_RULE_ORIGIN_MAX)]
    FW_RULE_ORIGIN_TYPE Origin;
[string, range(1, 10001)] wchar_t* wszGPONName;
unsigned long Reserved;
} FW_RULE2_0,
*PFW_RULE2_0;

```

2.2.36 FW_RULE

This structure is used to represent a firewall rule.

```

typedef struct _tag_FW_RULE {
    struct _tag_FW_RULE* pNext;
    unsigned short wSchemaVersion;
    [string, range(1, 10001), ref] wchar_t* wszRuleId;
    [string, range(1, 10001)] wchar_t* wszName;
    [string, range(1, 10001)] wchar_t* wszDescription;
    unsigned long dwProfiles;
    [range(FW_DIR_INVALID, FW_DIR_OUT)]
        FW_DIRECTION Direction;
    [range(0, 256)] unsigned short wIpProtocol;
    [switch_type(unsigned short), switch_is(wIpProtocol)]
        union {
            [case(6,17)]
                struct {
                    FW_PORTS LocalPorts;
                    FW_PORTS RemotePorts;
                };
            [case(1)]
                FW_ICMP_TYPE_CODE_LIST V4TypeCodeList;
            [case(58)]
                FW_ICMP_TYPE_CODE_LIST V6TypeCodeList;
            [default]
                ;
        };
    FW_ADDRESSES LocalAddresses;
    FW_ADDRESSES RemoteAddresses;
    FW_INTERFACE_LUIDS LocalInterfaceIds;
    unsigned long dwLocalInterfaceTypes;
    [string, range(1, 10001)] wchar_t* wszLocalApplication;
    [string, range(1, 10001)] wchar_t* wszLocalService;
    [range(FW_RULE_ACTION_INVALID, FW_RULE_ACTION_MAX)]
        FW_RULE_ACTION Action;
    unsigned short wFlags;
    [string, range(1, 10001)] wchar_t* wszRemoteMachineAuthorizationList;
    [string, range(1, 10001)] wchar_t* wszRemoteUserAuthorizationList;
    [string, range(1, 10001)] wchar_t* wszEmbeddedContext;
    FW_OS_PLATFORM_LIST PlatformValidityList;
    FW_RULE_STATUS Status;
    [range(FW_RULE_ORIGIN_INVALID, FW_RULE_ORIGIN_MAX)]
        FW_RULE_ORIGIN_TYPE Origin;
}

```

```

[string, range(1, 10001)] wchar_t* wszGPOName;
unsigned long Reserved;
[size_is((Reserved & FW_OBJECT_CTRL_FLAG_INCLUDE_METADATA) ? 1 : 0)]
    PFW_OBJECT_METADATA pMetaData;
} FW_RULE,
*PFW_RULE;

```

pNext: A pointer to the next **FW_RULE** in the list.

wSchemaVersion: Specifies the version of the rule.

wszRuleId: A pointer to a Unicode string that uniquely identifies the rule.

wszName: A pointer to a Unicode string that provides a friendly name for the rule.

wszDescription: A pointer to a Unicode string that provides a friendly description for the rule.

dwProfiles: A bitmask of the [FW_PROFILE_TYPE](#) flags. It is a condition that matches traffic on the specified profiles.

Direction: Specifies the direction of the traffic that the rule matches.

wIpProtocol: A condition that specifies the protocol of the traffic that the rule matches. If the value is within the range 0 to 255, the value describes a protocol in IETF IANA numbers (for more information, see [IANA-PROTO-NUM](#)). If the value is 256, the rule matches any protocol.

LocalPorts: A condition that specifies the local host ports of the TCP or UDP traffic that the rule matches.

RemotePorts: A condition that specifies the remote host ports of the TCP or UDP traffic that the rule matches.

V4TypeCodeList: A condition that specifies the list of ICMP types of the traffic that the rule matches. This field applies only when **wIpProtocol** specifies ICMP v4.

V6TypeCodeList: A condition that specifies the list of ICMP types of the traffic that the rule matches. This field applies only when **wIpProtocol** specifies ICMP v6.

LocalAddresses: A condition that specifies the addresses of the local host of the traffic that the rule matches. An empty **LocalAddresses** structure means that this condition is not applied.

RemoteAddresses: A condition that specifies the addresses of the remote host of the traffic that the rule matches. An empty **RemoteAddresses** structure means that this condition is not applied.

LocalInterfaceIds: A condition that specifies the list of specific network interfaces used by the traffic that the rule matches. A **LocalInterfaceIds** field with no interface GUID specified means that the rule applies to all interfaces; that is, the condition is not applied.

dwLocalInterfaceTypes: A bitmask of [FW_INTERFACE_TYPE](#). It is a condition that restricts the interface types that are used by the traffic that the rule matches. 0x00000000 means that the condition matches all interface types.

wszLocalApplication: A pointer to a Unicode string. It is a condition that specifies a file path name to the executable that uses the traffic that the rule matches. A null in this field means that the rule applies to all processes in the host.

wszLocalService: A pointer to a Unicode string. It is a condition that specifies the service name of the service that uses the traffic that the rule matches. An L"*" string in this field means that the rule applies to all services in the system. A null in this field means that the rule applies to all processes.

Action: The action that the rule will take for the traffic matches.

wFlags: Bit flags from [FW_RULE_FLAGS](#).

wszRemoteMachineAuthorizationList: A pointer to a Unicode string. A condition that specifies the remote machines sending or receiving the traffic that the rule matches. The string is in SDDL format ([\[MS-DTYP\]](#) section 2.5.1).

wszRemoteUserAuthorizationList: A pointer to a Unicode string. A condition that specifies the remote users accepting or receiving the traffic that the rule matches. The string is in SDDL format ([\[MS-DTYP\]](#) section 2.5.1).

wszEmbeddedContext: A pointer to a Unicode string. It specifies a group name for this rule. Other components in the system use this string to enable or disable groups of rules by verifying that they all have the same group name.

PlatformValidityList: A condition that specifies the platforms sending or receiving the traffic that the rule matches.

Status: The status code of the rule, as specified by the [FW_RULE_STATUS](#) enumeration. This field is filled out when the structure is returned as output. On input, this field MUST be set to FW_RULE_STATUS_OK.

Origin: The rule origin, as specified in the [FW_RULE_ORIGIN_TYPE](#) enumeration. It MUST be filled on enumerated rules and ignored on input.

wszGPOName: A Unicode string that represents the name of the originating GPO (see [\[MS-GPREG\]](#)). If the rule origin is not Group Policy, it MUST be NULL. Additionally, **wszGPOName** MUST only be set when retrieving rules through the [RRPC FWEnumFirewallRules](#) method if FW_ENUM_RULES_FLAG_RESOLVE_GPO_NAME is passed to the function.

Reserved: Not used other than to instruct RPC, using the FW_OBJECT_CTRL_FLAG_INCLUDE_METADATA flag, that a pointer to an FW_OBJECT_METADATA structure is present. It has no semantic meaning to the object itself.

pMetaData: A pointer to an FW_OBJECT_METADATA structure that contains specific metadata about the current state of the firewall rule.

The following are semantic checks that firewall rules MUST pass:

- The **wSchemaVersion** field MUST NOT be less than 0x000100.
- The **wSchemaVersion** field SHOULD NOT be less than 0x000200. [<7>](#)
- The **wszRuleId** field MUST NOT contain the pipe (|) character, MUST NOT be NULL, MUST be a string of at least 1 character, and MUST NOT be greater or equal to 512 characters. [<8>](#)
- The **wszName** field string MUST meet the following criteria:

- MUST contain 1 or more characters.
- MUST contain fewer than 10,000 characters.
- MUST NOT be NULL.
- MUST NOT contain the pipe (|) character.
- MUST NOT equal the case-insensitive string "ALL".
- If the **wszDescription** field string is not NULL, it MUST contain at least 1 character, MUST NOT be greater than or equal to 10,000 characters, and MUST NOT contain the pipe (|) character.
- If the **wszLocalApplication** field string is not NULL, it MUST be at least 1 character, MUST NOT be greater than or equal to MAX_PATH (260) characters, and MUST NOT contain the following characters: /,*,?,",<,>,|.
- If the **wszLocalService** field string is not NULL, it MUST contain at least 1 character, MUST NOT be greater than or equal to MAX_PATH characters, and MUST NOT contain the following characters: /,\,|.
- If the **wszEmbeddedContext** field string is not NULL, it MUST contain at least 1 character, MUST NOT be greater than or equal to 10,000 characters, and MUST NOT contain the pipe (|) character.
- The **Direction** field MUST NOT contain invalid [FW_DIRECTION](#) values.
- The **dwProfiles** field MUST NOT contain invalid values and, if it is not equal to the FW_PROFILE_TYPE_ALL profile type, it MUST NOT contain unknown profiles.
- The **wIpProtocol** field MUST NOT be greater than 256.
- If the **wPortKeywords** field of **LocalPorts** is FW_PORT_KEYWORD_DYNAMIC_RPC_PORTS or FW_PORT_KEYWORD_RPC_EP, the **wIpProtocol** field MUST be 6, and **Direction** MUST be FW_DIRECTION_IN.
- If the **wPortKeywords** field of **LocalPorts** is FW_PORT_KEYWORD_TEREDO_PORT, the **wIpProtocol** field MUST be 17, and **Direction** MUST be FW_DIRECTION_IN.
- The **wPortKeywords** field of **LocalPorts** MUST be 0 if the **Direction** is FW_DIRECTION_OUT.
- If the **wIpProtocol** field is 6 or 17, the **wPortKeywords** field of **RemotePorts** MUST be 0.
- If the **wIpProtocol** field is not 1, 6, 17, or 58, the **LocalPorts**, **RemotePorts**, **V4TypeCodeList**, and **V6TypeCodeList** field MUST be empty.
- The **dwV4AddressKeywords** and **dwV6AddressKeywords** fields of **LocalAddresses** MUST be 0.
- **dwLocalInterfaceTypes** MUST NOT be greater than or equal to FW_INTERFACE_TYPE_MAX.
- **Action** MUST be a valid action from the [FW_RULE_ACTION](#) enumeration.
- **wFlags** MUST NOT be greater than FW_RULE_FLAGS_MAX.
- If **Direction** is FW_DIR_OUT, **wFlags** MUST NOT contain a FW_RULE_FLAGS_ROUTEABLE_ADDRS_TRAVERSE.

- If **Direction** is FW_DIR_IN or **wIpProtocol** is 6 or **wFlags** contains FW_RULE_FLAGS_AUTHENTICATE or FW_RULE_FLAGS_AUTHENTICATE_WITH_ENCRYPTION, **wFlags** MUST NOT contain FW_RULE_FLAGS_LOOSE_SOURCE_MAPPED.
- The **wFlags** field MUST NOT contain both FW_RULE_FLAGS_AUTHENTICATE and FW_RULE_FLAGS_AUTHENTICATE_WITH_ENCRYPTION.
- If **wFlags** contains either FW_RULE_FLAGS_AUTHENTICATE or FW_RULE_FLAGS_AUTHENTICATE_WITH_ENCRYPTION, **Action** MUST NOT be FW_RULE_ACTION_BLOCK.
- If **Action** is FW_RULE_ACTION_ALLOW_BYPASS, **Direction** MUST be FW_DIR_IN, **wFlags** MUST contain either FW_RULE_FLAGS_AUTHENTICATE or FW_RULE_FLAGS_AUTHENTICATE_WITH_ENCRYPTION, and **wszRemoteMachineAuthorizationList** MUST NOT be NULL.
- If **wszRemoteMachineAuthorizationList** is not NULL, it MUST be at least 1 character, MUST NOT be greater than or equal to 10,000 characters, MUST NOT contain the pipe (|) character, MUST NOT be an empty string (""), MUST be a valid security descriptor ([\[MS-DTYP\]](#) section 2.4.6), MUST have a non-Null **ACL**, MUST have only either Allow or Deny **ACEs**, and each ACE MUST have a Filter match access right.
- If **wszRemoteUserAuthorizationList** is not NULL, it MUST be at least 1 character, MUST NOT be greater than or equal to 10,000 characters, MUST NOT contain the pipe (|) character, MUST NOT be an empty string (""), MUST be a valid security descriptor ([\[MS-DTYP\]](#) section 2.4.6), MUST have a non-NULL ACL, MUST only have either Allow or Deny ACes, and each ACE MUST have a Filter match access right.
- If **wszRemoteMachineAuthorizationList** is not NULL or **wszRemoteUserAuthorizationList** is not NULL, either the FW_RULE_FLAGS_AUTHENTICATE flag or the FW_RULE_FLAGS_AUTHENTICATE_WITH_ENCRYPT flag MUST be set on the **wFlags** field.
- If the **Direction** field is FW_DIR_OUT, the **wszRemoteMachineAuthorizationList** field MUST be NULL.

2.2.37 FW_PROFILE_CONFIG

This enumeration identifies each of the per-profile configuration options supported by this protocol. Each configuration option has a merge law that is used to determine how to merge the values of these options across stores.

```
typedef enum _tag_FW_PROFILE_CONFIG
{
    FW_PROFILE_CONFIG_INVALID,
    FW_PROFILE_CONFIG_ENABLE_FW,
    FW_PROFILE_CONFIG_DISABLE_STEALTH_MODE,
    FW_PROFILE_CONFIG_SHIELDED,
    FW_PROFILE_CONFIG_DISABLE_UNICAST_RESPONSES_TO_MULTICAST_BROADCAST,
    FW_PROFILE_CONFIG_LOG_DROPPED_PACKETS,
    FW_PROFILE_CONFIG_LOG_SUCCESS_CONNECTIONS,
    FW_PROFILE_CONFIG_LOG_IGNORED_RULES,
    FW_PROFILE_CONFIG_LOG_MAX_FILE_SIZE,
    FW_PROFILE_CONFIG_LOG_FILE_PATH,
    FW_PROFILE_CONFIG_DISABLE_INBOUND_NOTIFICATIONS,
    FW_PROFILE_CONFIG_AUTH_APPS_ALLOW_USER_PREF_MERGE,
    FW_PROFILE_CONFIG_GLOBAL_PORTS_ALLOW_USER_PREF_MERGE,
    FW_PROFILE_CONFIG_ALLOW_LOCAL_POLICY_MERGE,

```

```

FW_PROFILE_CONFIG_ALLOW_LOCAL_IPSEC_POLICY_MERGE,
FW_PROFILE_CONFIG_DISABLED_INTERFACES,
FW_PROFILE_CONFIG_DEFAULT_OUTBOUND_ACTION,
FW_PROFILE_CONFIG_DEFAULT_INBOUND_ACTION,
FW_PROFILE_CONFIG_MAX
} FW_PROFILE_CONFIG;

```

FW_PROFILE_CONFIG_INVALID: This value is invalid and MUST NOT be used. It is defined for simplicity in writing IDL definitions and code. This symbolic constant has a value of zero.

FW_PROFILE_CONFIG_ENABLE_FW: This value is an on/off switch for the firewall and advanced security enforcement. It is a **DWORD** type value; 0x00000000 is off; 0x00000001 is on. The merge law for this option is to let the value of the Group Policy RSOP store win if it is configured; otherwise, the local store value is used. This symbolic constant has a value of 1.

FW_PROFILE_CONFIG_DISABLE_STEALTH_MODE: This value is a **DWORD** used as an on/off switch for the stealth feature of firewall and advanced security. The merge law for this option is to let the value of the Group Policy RSOP store win if it is configured; otherwise, the local store value is used. This symbolic constant has a value of 2.

FW_PROFILE_CONFIG_SHIELDED: This value is a **DWORD** used as an on/off switch. If on, it blocks all incoming traffic regardless of what the rest of the policy specifies (except if the firewall is off). The merge law for this option is to let "on" values win. This symbolic constant has a value of 3.

FW_PROFILE_CONFIG_DISABLE_UNICAST_RESPONSES_TO_MULTICAST_BROADCAST: This value is a **DWORD** used as an on/off switch. If it is on, unicast responses to multicast broadcast traffic is blocked. The merge law for this option is to let the value of the Group Policy RSOP store win if it is configured; otherwise, the local store value is used. This symbolic constant has a value of 4.

FW_PROFILE_CONFIG_LOG_DROPPED_PACKETS: This value is a **DWORD** used as an on/off switch. If this value is on, the firewall logs all the dropped packets. The merge law for this option is to let "on" values win. This symbolic constant has a value of 5.

FW_PROFILE_CONFIG_LOG_SUCCESS_CONNECTIONS: This value is a **DWORD** used as an on/off switch. If this value is on, the firewall logs all successful inbound connections. The merge law for this option is to let "on" values win. This symbolic constant has a value of 6.

FW_PROFILE_CONFIG_LOG_IGNORED_RULES: This value is a **DWORD** used as an on/off switch. If this value is on, the host enforcing this policy logs events for each ignored rule in the policy. The merge law for this option is to let "on" values win. This symbolic constant has a value of 7.

FW_PROFILE_CONFIG_LOG_MAX_FILE_SIZE: This value is a **DWORD** and specifies the size, in kilobytes, of the log where dropped packets and successful connections are logged. The merge law for this option is to let the value of the Group Policy RSOP store win if it is configured; otherwise, the local store value is used. This symbolic constant has a value of 8.

FW_PROFILE_CONFIG_LOG_FILE_PATH: This configuration value is a string that represents a file path to the log for when the firewall logs dropped packets and successful connections. The merge law for this option is to let the value of the Group Policy RSOP store win if it is configured; otherwise, the local store value is used. This symbolic constant has a value of 9.

FW_PROFILE_CONFIG_DISABLE_INBOUND_NOTIFICATIONS: This value is a **DWORD** used as an on/off switch. If this value is off, whenever an unknown application listens on a

port, the firewall displays a notification to the user who ran the application, prompting for an action. The merge law for this option is to let the value of the Group Policy RSOP store win if it is configured; otherwise, the local store value is used. This symbolic constant has a value of 10.

FW_PROFILE_CONFIG_AUTH_APPS_ALLOW_USER_PREF_MERGE: This value is a **DWORD** used as an on/off switch. If this value is off, authorized application settings (from the legacy Windows Firewall policy) in the local store are ignored and not enforced. The merge law for this option is to let the value of the Group Policy RSOP store win if it is configured; otherwise, the local store value is used. This symbolic constant has a value of 11.

FW_PROFILE_CONFIG_GLOBAL_PORTS_ALLOW_USER_PREF_MERGE: This value is a **DWORD** used as an on/off switch. If this value is off, global open port settings (from the legacy Windows Firewall policy) in the local store are ignored and not enforced. The setting only has meaning if it is set or enumerated in the Group Policy store or if it is enumerated from the Group Policy RSOP store. The merge law for this option is to let the value Group Policy RSOP store win if it is configured; otherwise, the local store value is used. This symbolic constant has a value of 12.

FW_PROFILE_CONFIG_ALLOW_LOCAL_POLICY_MERGE: This value is a **DWORD** used as an on/off switch. If this value is off, firewall rules (regardless of their schema version and firewall version) from the local store are ignored and not enforced. The merge law for this option is to always use the value of the Group Policy RSOP store. This symbolic constant has a value of 13.

FW_PROFILE_CONFIG_ALLOW_LOCAL_IPSEC_POLICY_MERGE: This value is a **DWORD**; it is an on/off switch. If this value is off, connection security rules from the local store are ignored and not enforced, regardless of the schema version and connection security rule version. The merge law for this option is to always use the value of the Group Policy RSOP store. This symbolic constant has a value of 14.

FW_PROFILE_CONFIG_DISABLED_INTERFACES: This value is an [FW_INTERFACE_LUIDS](#) structure that represents the network adapters where the firewall (only the firewall rules and actions) is off. The merge law for this option is to let the value of the Group Policy RSOP store win if it is configured; otherwise, the local store value is used. This symbolic constant has a value of 15.

FW_PROFILE_CONFIG_DEFAULT_OUTBOUND_ACTION: This value is the action that the firewall does by default (and evaluates at the very end) on outbound connections. The allow action is represented by 0x00000000; 0x00000001 represents a block action. The merge law for this option is to let the value of the Group Policy RSOP store win if it is configured; otherwise, the local store value is used. This symbolic constant has a value of 16.

FW_PROFILE_CONFIG_DEFAULT_INBOUND_ACTION: This value is the action that the firewall does by default (and evaluates at the very end) on inbound connections. The allow action is represented by 0x00000000; 0x00000001 represents a block action. The merge law for this option is to let the value of the Group Policy RSOP store win if it is configured; otherwise, the local store value is used. This symbolic constant has a value of 17.

FW_PROFILE_CONFIG_MAX: This value is invalid. It is defined for simplicity in writing IDL definitions and code. Values greater than this value are also not valid. This symbolic constant has a value of 18.

2.2.38 FW_GLOBAL_CONFIG_IPSEC_EXEMPT_VALUES

This enumeration identifies specific traffic to be exempted from performing IPsec.

```

typedef enum _FW_GLOBAL_CONFIG_IPSEC_EXEMPT_VALUES
{
    FW_GLOBAL_CONFIG_IPSEC_EXEMPT_NONE = 0x0000,
    FW_GLOBAL_CONFIG_IPSEC_EXEMPT_NEIGHBOR_DISC = 0x0001,
    FW_GLOBAL_CONFIG_IPSEC_EXEMPT_ICMP = 0x0002,
    FW_GLOBAL_CONFIG_IPSEC_EXEMPT_ROUTER_DISC = 0x0004,
    FW_GLOBAL_CONFIG_IPSEC_EXEMPT_DHCP = 0x0008,
    FW_GLOBAL_CONFIG_IPSEC_EXEMPT_MAX = 0x0010,
    FW_GLOBAL_CONFIG_IPSEC_EXEMPT_MAX_V2_0 = 0x0004
} FW_GLOBAL_CONFIG_IPSEC_EXEMPT_VALUES;

```

FW_GLOBAL_CONFIG_IPSEC_EXEMPT_NONE: No IPsec exemptions.

FW_GLOBAL_CONFIG_IPSEC_EXEMPT_NEIGHBOR_DISC: Exempt neighbor discover IPv6 ICMP type-codes from IPsec.

FW_GLOBAL_CONFIG_IPSEC_EXEMPT_ICMP: Exempt ICMP from IPsec.

FW_GLOBAL_CONFIG_IPSEC_EXEMPT_ROUTER_DISC: Exempt router discover IPv6 ICMP type-codes from IPsec.

FW_GLOBAL_CONFIG_IPSEC_EXEMPT_DHCP: Exempt both IPv4 and IPv6 DHCP traffic from IPsec.

FW_GLOBAL_CONFIG_IPSEC_EXEMPT_MAX: This value and greater values MUST NOT be used. It is defined for simplicity in writing IDL definitions and code.

FW_GLOBAL_CONFIG_IPSEC_EXEMPT_MAX_V2_0: For schema version 0x0200, this value and greater values are invalid and MUST NOT be used. This symbolic constant is defined for simplicity in writing IDL definitions and describing semantic checks against policy schema versions of 0x0200.

2.2.39 FW_GLOBAL_CONFIG_PRESHARED_KEY_ENCODING_VALUES

This enumeration is used to describe how preshared keys are encoded before being used.

```

typedef enum _FW_GLOBAL_CONFIG_PRESHARED_KEY_ENCODING_VALUES
{
    FW_GLOBAL_CONFIG_PRESHARED_KEY_ENCODING_NONE = 0,
    FW_GLOBAL_CONFIG_PRESHARED_KEY_ENCODING_UTF_8,
    FW_GLOBAL_CONFIG_PRESHARED_KEY_ENCODING_MAX
} FW_GLOBAL_CONFIG_PRESHARED_KEY_ENCODING_VALUES;

```

FW_GLOBAL_CONFIG_PRESHARED_KEY_ENCODING_NONE: Preshared key is not encoded. Instead, it is kept in its wide-character format. This symbolic constant has a value of zero.

FW_GLOBAL_CONFIG_PRESHARED_KEY_ENCODING_UTF_8: Encode the preshared key using UTF-8. This symbolic constant has a value of 1.

FW_GLOBAL_CONFIG_PRESHARED_KEY_ENCODING_MAX: This value and greater values are invalid. This value is defined for simplicity in writing IDL definitions and code. This symbolic constant has a value of 2.

2.2.40 FW_GLOBAL_CONFIG_IPSEC_THROUGH_NAT_VALUES

This enumeration is used to describe when IPsec security associations can be established across NAT devices.

```
typedef enum _FW_GLOBAL_CONFIG_IPSEC_THROUGH_NAT_VALUES
{
    FW_GLOBAL_CONFIG_IPSEC_THROUGH_NAT_NEVER = 0,
    FW_GLOBAL_CONFIG_IPSEC_THROUGH_NAT_SERVER_BEHIND_NAT,
    FW_GLOBAL_CONFIG_IPSEC_THROUGH_NAT_SERVER_AND_CLIENT_BEHIND_NAT,
    FW_GLOBAL_CONFIG_IPSEC_THROUGH_NAT_MAX
} FW_GLOBAL_CONFIG_IPSEC_THROUGH_NAT_VALUES;
```

FW_GLOBAL_CONFIG_IPSEC_THROUGH_NAT_NEVER: IPsec does not cross NAT boundaries. This symbolic constant has a value of zero.

FW_GLOBAL_CONFIG_IPSEC_THROUGH_NAT_SERVER_BEHIND_NAT: IPsec security associations can be established when the server is across NAT boundaries. This symbolic constant has a value of 1.

FW_GLOBAL_CONFIG_IPSEC_THROUGH_NAT_SERVER_AND_CLIENT_BEHIND_NAT: IPsec security associations can be established when the server and client are across NAT boundaries. This symbolic constant has a value of 2.

FW_GLOBAL_CONFIG_IPSEC_THROUGH_NAT_MAX: This value and greater values are invalid and MUST NOT be used. It is defined for simplicity in writing IDL definitions and code. This symbolic constant has a value of 3.

2.2.41 FW_GLOBAL_CONFIG

This enumeration identifies the global policy configuration options. Each configuration option has a merge law that is used to determine how to merge the values of these options across stores.

```
typedef enum _tag_FW_GLOBAL_CONFIG
{
    FW_GLOBAL_CONFIG_INVALID,
    FW_GLOBAL_CONFIG_POLICY_VERSION_SUPPORTED,
    FW_GLOBAL_CONFIG_CURRENT_PROFILE,
    FW_GLOBAL_CONFIG_DISABLE_STATEFUL_FTP,
    FW_GLOBAL_CONFIG_DISABLE_STATEFUL_PPTP,
    FW_GLOBAL_CONFIG_SA_IDLE_TIME,
    FW_GLOBAL_CONFIG_PRESHARED_KEY_ENCODING,
    FW_GLOBAL_CONFIG_IPSEC_EXEMPT,
    FW_GLOBAL_CONFIG_CRL_CHECK,
    FW_GLOBAL_CONFIG_IPSEC_THROUGH_NAT,
    FW_GLOBAL_CONFIG_POLICY_VERSION,
    FW_GLOBAL_CONFIG_BINARY_VERSION_SUPPORTED,
    FW_GLOBAL_CONFIG_IPSEC_TUNNEL_REMOTE_MACHINE_AUTHORIZATION_LIST,
    FW_GLOBAL_CONFIG_IPSEC_TUNNEL_REMOTE_USER_AUTHORIZATION_LIST,
    FW_GLOBAL_CONFIG_MAX
} FW_GLOBAL_CONFIG;
```

FW_GLOBAL_CONFIG_INVALID: This value MUST NOT be used. It is defined for simplicity in writing IDL definitions and code. This symbolic constant has a value of zero.

FW_GLOBAL_CONFIG_POLICY_VERSION_SUPPORTED: This value is a **DWORD** containing the maximum policy version that the server host can accept. The version number is two octets in size. The lowest-order octet is the minor version; the second-to-lowest octet is the major version. This value is not merged and is always a fixed value for a particular firewall and advanced security components software build. This symbolic constant has a value of 1.

FW_GLOBAL_CONFIG_CURRENT_PROFILE: This value is a **DWORD** and contains a bitmask of the current enforced profiles that are maintained by the server firewall host. This value is available only in the dynamic store; therefore, it is not merged and has no merge law. This symbolic constant has a value of 2.

FW_GLOBAL_CONFIG_DISABLE_STATEFUL_FTP: This value is an on/off switch. If off, the firewall performs stateful File Transfer Protocol (FTP) filtering to allow secondary connections. The value is a **DWORD**; 0x00000000 means off; 0x00000001 means on. The merge law for this option is to let "on" values win. This symbolic constant has a value of 3.

FW_GLOBAL_CONFIG_DISABLE_STATEFUL_PPTP: This value is an on/off switch. If off, the firewall performs stateful Point-to-Point Tunneling Protocol (PPTP) analysis. The value is a **DWORD**; 0x00000000 means off; 0x00000001 means on. The merge law for this option is to let "on" values win. This symbolic constant has a value of 4.

FW_GLOBAL_CONFIG_SA_IDLE_TIME: This value configures the security association idle time, in seconds. Security associations are deleted after network traffic is not seen for this specified period of time. The value is a **DWORD** and MUST be a value in the range of 300 to 3,600 inclusive. The merge law for this option is to let the value of the Group Policy RSOP store win if it is configured; otherwise, use the local store value. This symbolic constant has a value of 5.

FW_GLOBAL_CONFIG_PRESHARED_KEY_ENCODING: This configuration value specifies the preshared key encoding that is used. The value is a **DWORD** and MUST be a valid value from the [FW_GLOBAL_CONFIG_PRESHARED_KEY_ENCODING_VALUES](#) enumeration. The merge law for this option is to let the value of the Group Policy RSOP store win if it is configured; otherwise, use the local store value. This symbolic constant has a value of 6.

FW_GLOBAL_CONFIG_IPSEC_EXEMPT: This configuration value configures IPsec exceptions. The value is a **DWORD** and MUST be a combination of the valid flags that are defined in [FW_GLOBAL_CONFIG_IPSEC_EXEMPT_VALUES](#); therefore, the maximum value MUST always be FW_GLOBAL_CONFIG_IPSEC_EXEMPT_MAX-1 for servers supporting a schema version of 0x0201 and FW_GLOBAL_CONFIG_IPSEC_EXEMPT_MAX_V2_0-1 for servers supporting a schema version of 0x0200. If the maximum value is exceeded when the method [RRPC FWSetGlobalConfig \(Opnum 4\)](#) is called, the method returns ERROR_INVALID_PARAMETER. This error code is returned if no other preceding error is discovered. The merge law for this option is to let the value of the Group Policy RSOP store win if it is configured; otherwise, use the local store value. This symbolic constant has a value of 7.

FW_GLOBAL_CONFIG_CRL_CHECK: This value specifies how **certificate revocation list (CRL)** verification is enforced. The value is a **DWORD** and MUST be 0, 1, or 2. A value of 0 disables CRL checking. A value of 1 specifies that CRL checking is attempted and that certificate validation fails only if the certificate is revoked. Other failures that are encountered during CRL checking (such as the revocation URL being unreachable) do not cause certificate validation to fail. A value of 2 means that checking is required and that certificate validation fails if any error is encountered during CRL processing. The merge law for this option is to let the value of the Group Policy RSOP store win if it is configured; otherwise, use the local store value. This symbolic constant has a value of 8.

FW_GLOBAL_CONFIG_IPSEC_THROUGH_NAT: This value is configured when an IPsec security association can be established with a computer across NAT devices. The value is of type [FW_GLOBAL_CONFIG_IPSEC_THROUGH_NAT_VALUES](#) and MUST contain valid values of the same enumeration type. The merge law for this option is to let the value of the Group Policy RSOP store win if it is configured; otherwise, use the local store value. This symbolic constant has a value of 9.

FW_GLOBAL_CONFIG_POLICY_VERSION: This value contains the policy version of the policy store being managed. This value is not merged and therefore, has no merge law. This symbolic constant has a value of 10.

FW_GLOBAL_CONFIG_BINARY_VERSION_SUPPORTED: This value contains the binary version of the structures and data types that are supported by the server. This value is not merged. In addition, this value is always a fixed value for a specific firewall and advanced security component's software build. This symbolic constant has a value of 11. This value identifies a policy configuration option that is supported only on servers that have a schema version of 0x0201.

FW_GLOBAL_CONFIG_IPSEC_TUNNEL_REMOTE_MACHINE_AUTHORIZATION_LIST: This value represents a list of remote machines that are allowed to send and receive traffic through the tunnels which request this access check. Machines in the list are allowed through the tunnels. Machines not in the list are denied through the tunnels. The list is specified as a security descriptor which specifies which SIDs ([\[MS-DTYP\]](#) section 2.4.2.1) of the remote machines. The value is a Unicode string in Security Descriptor Definition Language (SDDL) format ([\[MS-DTYP\]](#) section 2.5.1). This symbolic constant has a value of 12.

FW_GLOBAL_CONFIG_IPSEC_TUNNEL_REMOTE_USER_AUTHORIZATION_LIST: This value represents a list of remote users who are allowed to send and receive traffic through the tunnels which request this access check. Users in the list are allowed through the tunnels. Users not in the list are denied through the tunnels. The list is specified as a security descriptor which specifies which SIDs ([\[MS-DTYP\]](#) section 2.4.2.1) of the remote users. The value is a Unicode string in SDDL format ([\[MS-DTYP\]](#) section 2.5.1). This symbolic constant has a value of 13.

FW_GLOBAL_CONFIG_MAX: This value and values that exceed this value are not valid and MUST NOT be used. This symbolic constant is defined for simplicity in writing IDL definitions and code. It has a value of 14.

2.2.42 FW_CONFIG_FLAGS

This enumeration identifies flags that can be set on the [RRPC_FWGetConfig \(Opnum 10\)](#) and [RRPC_FWGetGlobalConfig \(Opnum 3\)](#) methods.

```
typedef enum _FW_CONFIG_FLAGS
{
    FW_CONFIG_FLAG_RETURN_DEFAULT_IF_NOT_FOUND = 0x0001
} FW_CONFIG_FLAGS;
```

FW_CONFIG_FLAG_RETURN_DEFAULT_IF_NOT_FOUND: If this flag is specified, and if the [RRPC_FWGetConfig \(Opnum 10\)](#) method or the [RRPC_FWGetGlobalConfig \(Opnum 3\)](#) method fails to find the configuration value in the policy store, then the call will succeed and return the default value used by the firewall service. If this flag is not specified, these methods will fail with [ERROR_FILE_NOT_FOUND](#). The default set of values returned by these two calls is a firewall and advanced security component implementation-specific [<9>](#) decision, and is outside the scope of this protocol specification.

2.2.43 FW_NETWORK

This structure represents a network that is associated with a firewall profile. It is used for display purposes in user interfaces.

```
typedef struct _tag_FW_NETWORK {
    [string, unique] wchar_t* pszName;
    FW_PROFILE_TYPE ProfileType;
} FW_NETWORK,
*PFW_NETWORK;
```

pszName: A pointer to a Unicode string that represents the name of the network.

ProfileType: The profile type that is associated with the network. The type MUST be one of the [FW_PROFILE_TYPE](#) flags, except FW_PROFILE_TYPE_ALL.

2.2.44 FW_ADAPTER

This structure represents a network interface in the host. It is used for display purposes in the user interface when configuring the **FW_PROFILE_CONFIG_DISABLED_INTERFACES** (section [2.2.37](#)) configuration option.

```
typedef struct _tag_FW_ADAPTER {
    [string, unique] wchar_t* pszFriendlyName;
    GUID Guid;
} FW_ADAPTER,
*PFW_ADAPTER;
```

pszFriendlyName: A pointer to a Unicode string that presents the friendly name that is associated with the network interface.

Guid: A GUID that uniquely identifies the interface in the host system.

2.2.45 FW_DIAG_APP

This structure is not used on the wire.

2.2.46 FW_RULE_CATEGORY

This enumeration represents the classes of functionality that a third-party software component can register for, take ownership of, and commit to implement. The implementation of such functionality by the firewall and advanced security component, or by the third-party software component, are implementation-specific decisions. This enumeration is only used to present the state of the registrations.

```
typedef [vl_enum] enum _tag_FW_RULE_CATEGORY
{
    FW_RULE_CATEGORY_BOOT,
    FW_RULE_CATEGORY_STEALTH,
    FW_RULE_CATEGORY_FIREWALL,
    FW_RULE_CATEGORY_CONSEC,
    FW_RULE_CATEGORY_MAX
}
```



```
} FW_RULE_CATEGORY,  
 *PFW_RULE_CATEGORY;
```

FW_RULE_CATEGORY_BOOT: This category of functionality represents the policy that is used while the system is starting up and the firewall and advanced security component is not yet running. This symbolic constant has a value of 0.

FW_RULE_CATEGORY_STEALTH: This category of functionality represents the policy that is used to make the system appear invisible when it is connected to a network. For example, this functionality helps prevent attackers from discovering the host and the ports that open to the host. This symbolic constant has a value of 1.

FW_RULE_CATEGORY_FIREWALL: This category of functionality represents functions that are performed by firewall objects while they are present on the FW_STORE_TYPE_LOCAL, FW_STORE_TYPE_DYNAMIC, and FW_STORE_TYPE_GP_RSOP policy stores (see section [2.2.1](#)). This symbolic constant has a value of 2.

FW_RULE_CATEGORY_CONSEC: This category of functionality represents functions that are performed by the connection security objects. This symbolic constant has a value of 3.

FW_RULE_CATEGORY_MAX: This value and greater values are invalid and MUST NOT be used. This symbolic constant is defined for simplicity in writing IDL definitions and code. It has a value of 4.

2.2.47 FW_PRODUCT

This structure represents a third-party software component that registers with the firewall and advanced security component to implement some of the categories.

```
typedef struct _tag_FW_PRODUCT {  
    DWORD dwFlags;  
    DWORD dwNumRuleCategories;  
    [size_is(dwNumRuleCategories), unique]  
    FW_RULE_CATEGORY* pRuleCategories;  
    [string, ref] wchar_t* pszDisplayName;  
    [string, unique] wchar_t* pszPathToSignedProductExe;  
} FW_PRODUCT,  
 *PFW_PRODUCT;
```

dwFlags: This field is not used.

dwNumRuleCategories: The number of rule categories with which the third-party software component registered.

pRuleCategories: A pointer to an array of **dwNumRuleCategories** that are contiguous [FW_RULE_CATEGORY](#) elements.

pszDisplayName: A pointer to a Unicode string. The string represents the name of the third-party software component.

pszPathToSignedProductExe: A pointer to a Unicode string. The string represents the file path to the binary executable of the third-party software component.

2.2.48 FW_IP_VERSION

This enumeration is used to represent the two current IP protocol versions in use: IP version 4 and IP version 6.

```
typedef enum _tag_FW_IP_VERSION
{
    FW_IP_VERSION_INVALID,
    FW_IP_VERSION_V4,
    FW_IP_VERSION_V6,
    FW_IP_VERSION_MAX
} FW_IP_VERSION;
```

FW_IP_VERSION_INVALID: This value MUST NOT be used. It is defined for simplicity in writing IDL definitions and code. This symbolic constant has a value of zero.

FW_IP_VERSION_V4: This value represents IPv4. This symbolic constant has a value of 1.

FW_IP_VERSION_V6: This value represents the IPv6. This symbolic constant has a value of 2.

FW_IP_VERSION_MAX: This value and greater values are invalid and MUST NOT be used. It is defined for simplicity in writing IDL definitions and code. This symbolic constant has a value of 3.

2.2.49 FW_IPSEC_PHASE

This enumeration is used to identify the IPsec phase of negotiations.

```
typedef enum _tag_FW_IPSEC_PHASE
{
    FW_IPSEC_PHASE_INVALID,
    FW_IPSEC_PHASE_1,
    FW_IPSEC_PHASE_2,
    FW_IPSEC_PHASE_MAX
} FW_IPSEC_PHASE;
```

FW_IPSEC_PHASE_INVALID: This value MUST NOT be used. It is defined for simplicity in writing IDL definitions and code. This symbolic constant has a value of zero.

FW_IPSEC_PHASE_1: This value represents the IPsec first phase of negotiations, also called main mode. This symbolic constant has a value of 1.

FW_IPSEC_PHASE_2: This value represents the IPsec second phase of negotiations. A phase 2 authentication is the second authentication and can mean extended mode or quick mode. On auth sets, phase 2 authentication refers to extended mode (specified in [\[MS-AIPS\]](#) sections [3.6](#) and [3.7](#)); and on crypto sets, phase 2 refers to quick mode (specified in [\[MS-AIPS\]](#) sections [3.4](#) and [3.5](#)). This symbolic constant has a value of 2.

FW_IPSEC_PHASE_MAX: This value and greater values are invalid and MUST NOT be used. It is defined for simplicity in writing IDL definitions and code. This symbolic constant has a value of 3.

2.2.50 FW_CS_RULE_FLAGS

This enumeration describes flag values for connection security rules.

```

typedef enum _tag_FW_CS_RULE_FLAGS
{
    FW_CS_RULE_FLAGS_NONE = 0x00,
    FW_CS_RULE_FLAGS_ACTIVE = 0x01,
    FW_CS_RULE_FLAGS_DTM = 0x02,
    FW_CS_RULE_TUNNEL_BYPASS_IF_ENCRYPTED = 0x08,
    FW_CS_RULE_OUTBOUND_CLEAR = 0x10,
    FW_CS_RULE_FLAGS_APPLY_AUTHZ = 0x20,
    FW_CS_RULE_FLAGS_MAX = 0x40,
    FW_CS_RULE_FLAGS_MAX2_1 = 0x02
} FW_CS_RULE_FLAGS;

```

FW_CS_RULE_FLAGS_NONE: This value means that none of the following flags are set. This value is defined for simplicity in writing IDL definitions and code.

FW_CS_RULE_FLAGS_ACTIVE: If this flag is set, the rule is enabled; otherwise, the rule is disabled.

FW_CS_RULE_FLAGS_DTM: If this flag is set, the rule is a dynamic tunnel mode rule.

FW_CS_RULE_TUNNEL_BYPASS_IF_ENCRYPTED: This flag MUST only be set on tunnel mode rules. If this flag is set and traffic is already arriving encrypted, it is exempted from the tunnel.

FW_CS_RULE_OUTBOUND_CLEAR: This flag MUST only be set on tunnel mode rules. If set, when outbound traffic matches the rule, it leaves unprotected, but inbound traffic MUST arrive through the tunnel.

FW_CS_RULE_FLAGS_APPLY_AUTHZ: This flag MUST only be set on tunnel mode rules. If this flag is set, the authenticated peers of the traffic MUST match the SDDLs that are specified in FW_GLOBAL_CONFIG_IPSEC_TUNNEL_REMOTE_MACHINE_AUTHORIZATION_LIST and FW_GLOBAL_CONFIG_IPSEC_TUNNEL_REMOTE_USER_AUTHORIZATION_LIST.

FW_CS_RULE_FLAGS_MAX: This value and greater values are invalid and MUST NOT be used. This value is only defined for simplicity in writing IDL definitions and code.

FW_CS_RULE_FLAGS_MAX2_1: For schema version 0x0210, this value and values that exceed this value are not valid and MUST NOT be used. This value is only defined for simplicity in writing IDL definitions and code.

2.2.51 FW_CS_RULE_ACTION

This enumeration identifies the possible actions a connection security rule (section [2.2.53](#)) can have.

```

typedef enum _tag_FW_CS_RULE_ACTION
{
    FW_CS_RULE_ACTION_INVALID,
    FW_CS_RULE_ACTION_SECURE_SERVER,
    FW_CS_RULE_ACTION_BOUNDARY,
    FW_CS_RULE_ACTION_SECURE,
    FW_CS_RULE_ACTION_DO_NOT_SECURE,
    FW_CS_RULE_ACTION_MAX
} FW_CS_RULE_ACTION;

```

FW_CS_RULE_ACTION_INVALID: This value MUST NOT be used. It is defined for simplicity in writing IDL definitions and code. This symbolic constant has a value of zero.

FW_CS_RULE_ACTION_SECURE_SERVER: This action requires inbound traffic to be IPsec traffic and attempts to secure outbound traffic with IPsec. This symbolic constant has a value of 1.

FW_CS_RULE_ACTION_BOUNDARY: This action attempts to secure inbound and outbound traffic with IPsec. If the action fails to secure the traffic, the traffic still flows on the clear. This symbolic constant has a value of 2.

FW_CS_RULE_ACTION_SECURE: This action requires inbound and outbound traffic to be secured by IPsec. This symbolic constant has a value of 3.

FW_CS_RULE_ACTION_DO_NOT_SECURE: This action exempts the traffic from being secured by IPsec. This symbolic constant has a value of 4.

FW_CS_RULE_ACTION_MAX: This value and greater values are invalid and MUST NOT be used. It is defined for simplicity in writing IDL definitions and code. This symbolic constant has a value of 5.

2.2.52 FW_CS_RULE2_0

This structure describes a connection security rule that is used by the 2.0 binary version for servers and clients (see sections [1.7](#) and [2.2.37](#)). The fields of this structure are identical to the [FW_RULE](#) structure and their meanings are covered in section [2.2.53](#).

```
typedef struct _tag_FW_CS_RULE2_0 {
    struct _tag_FW_CS_RULE2_0* pNext;
    unsigned short wSchemaVersion;
    [string, range(1,10001), ref] wchar_t* wszRuleId;
    [string, range(1,10001)] wchar_t* wszName;
    [string, range(1,10001)] wchar_t* wszDescription;
    unsigned long dwProfiles;
    FW_ADDRESSES Endpoint1;
    FW_ADDRESSES Endpoint2;
    FW_INTERFACE_LUIDS LocalInterfaceIds;
    unsigned long dwLocalInterfaceTypes;
    unsigned long dwLocalTunnelEndpointV4;
    unsigned char LocalTunnelEndpointV6[16];
    unsigned long dwRemoteTunnelEndpointV4;
    unsigned char RemoteTunnelEndpointV6[16];
    FW_PORTS Endpoint1Ports;
    FW_PORTS Endpoint2Ports;
    [range(0,256)] unsigned short wIpProtocol;
    [string, range(1,10001)] wchar_t* wszPhase1AuthSet;
    [string, range(1,10001)] wchar_t* wszPhase2CryptoSet;
    [string, range(1,10001)] wchar_t* wszPhase2AuthSet;
    [range(FW_CS_RULE_ACTION_SECURE_SERVER, FW_CS_RULE_ACTION_MAX)]
        FW_CS_RULE_ACTION Action;
    unsigned short wFlags;
    [string, range(1,10001)] wchar_t* wszEmbeddedContext;
    FW_OS_PLATFORM_LIST PlatformValidityList;
    [range(FW_RULE_ORIGIN_INVALID, FW_RULE_ORIGIN_MAX-1)]
        FW_RULE_ORIGIN_TYPE Origin;
    [string, range(1,10001)] wchar_t* wszGPOName;
    FW_RULE_STATUS Status;
} FW_CS_RULE2_0,
```

```
*PFW_CS_RULE2_0;
```

2.2.53 FW_CS_RULE

This structure describes a connection security rule.

```
typedef struct _tag_FW_CS_RULE {
    struct _tag_FW_CS_RULE* pNext;
    unsigned short wSchemaVersion;
    [string, range(1,10001), ref] wchar_t* wszRuleId;
    [string, range(1,10001)] wchar_t* wszName;
    [string, range(1,10001)] wchar_t* wszDescription;
    unsigned long dwProfiles;
    FW_ADDRESSES Endpoint1;
    FW_ADDRESSES Endpoint2;
    FW_INTERFACE_LUIDS LocalInterfaceIds;
    unsigned long dwLocalInterfaceTypes;
    unsigned long dwLocalTunnelEndpointV4;
    unsigned char LocalTunnelEndpointV6[16];
    unsigned long dwRemoteTunnelEndpointV4;
    unsigned char RemoteTunnelEndpointV6[16];
    FW_PORTS Endpoint1Ports;
    FW_PORTS Endpoint2Ports;
    [range(0,256)] unsigned short wIpProtocol;
    [string, range(1,10001)] wchar_t* wszPhase1AuthSet;
    [string, range(1,10001)] wchar_t* wszPhase2CryptoSet;
    [string, range(1,10001)] wchar_t* wszPhase2AuthSet;
    [range(FW_CS_RULE_ACTION_SECURE_SERVER, FW_CS_RULE_ACTION_MAX)]
    FW_CS_RULE_ACTION Action;
    unsigned short wFlags;
    [string, range(1,10001)] wchar_t* wszEmbeddedContext;
    FW_OS_PLATFORM_LIST PlatformValidityList;
    [range(FW_RULE_ORIGIN_INVALID, FW_RULE_ORIGIN_MAX-1)]
    FW_RULE_ORIGIN_TYPE Origin;
    [string, range(1,10001)] wchar_t* wszGPOName;
    FW_RULE_STATUS Status;
    [string, range(1,512)] WCHAR* wszMMParentRuleId;
    DWORD Reserved;
    [size_is((Reserved & FW_OBJECT_CTRL_FLAG_INCLUDE_METADATA) ? 1 : 0)]
    PFW_OBJECT_METADATA pMetaData;
} FW_CS_RULE,
*PFW_CS_RULE;
```

pNext: A pointer to the next **FW_CS_RULE** in the list.

wSchemaVersion: Specifies the version of the rule.

wszRuleId: A pointer to a Unicode string that uniquely identifies the rule.

wszName: A pointer to a Unicode string that provides a friendly name for the rule.

wszDescription: A pointer to a Unicode string that provides a friendly description for the rule.

dwProfiles: A bitmask of the [FW_PROFILE_TYPE](#) flags. It is a condition that matches traffic on the specified profiles.

EndPoint1: A condition that specifies the addresses of the first host of the traffic that the rule matches. An empty **EndPoint1** structure means that this condition is not applied (any match).

EndPoint2: A condition that specifies the addresses of the second host of the traffic that the rule matches. An empty **EndPoint2** structure means that this condition is not applied (any match).

LocalInterfaceIds: A condition that specifies the list of specific network interfaces that are used by the traffic that the rule matches. If the **LocalInterfaceIds** field does not specify an interface GUID, the rule applies to all interfaces; that is, the condition is not applied.

dwLocalInterfaceTypes: A bitmask of [FW_INTERFACE_TYPE](#). It is a condition that restricts the interface types used by the traffic that the rule matches. A value of 0x00000000 means the condition matches all interface types.

dwLocalTunnelEndpointV4: This field specifies the IPv4 address of the endpoint that the host machines use as their local endpoint when IPsec operates in tunnel mode.

LocalTunnelEndpointV6: This field specifies the IPv6 address of the endpoint that the host machines use as their local endpoint when IPsec operates in tunnel mode.

dwRemoteTunnelEndpointV4: This field specifies the IPv4 address of the endpoint that the host machines use as their remote endpoint when IPsec operates in tunnel mode.

RemoteTunnelEndpointV6: This field specifies the IPv6 address of the endpoint that the host machines use as their remote endpoint when IPsec operates in tunnel mode.

EndPoint1Ports: A condition that specifies the first host's ports of the TCP or UDP traffic that the rule matches.

EndPoint2Ports: A condition that specifies the second host's ports of the TCP or UDP traffic that the rule matches.

wIpProtocol: A condition that specifies the protocol of the traffic that the rule matches. If the value is in the range of 0 to 255, the value describes a protocol as in IETF IANA numbers (for more information, see [IANA-PROTO-NUM](#)). If the value is 256, the rule matches any protocol.

wszPhase1AuthSet: A Unicode string that represents the set identifier for the Phase1 authentication policy objects.

wszPhase2CryptoSet: A Unicode string that represents the set identifier for the Phase2 cryptographic policy objects.

wszPhase2AuthSet: A Unicode string that represents the set identifier of the Phase2 authentication policy objects. If this field is NULL, no second authentication is performed.

Action: The connection security action that the rule takes for the traffic matches. This field MUST contain a valid value from the [FW_CS_RULE_ACTION](#) enumeration.

wFlags: A bit flag or flags from [FW_CS_RULE_FLAGS](#).

wszEmbeddedContext: A pointer to a Unicode string. It specifies a group name for this rule. Other components in the system use this string to enable or disable a group of rules by verifying that all rules have the same group name.

PlatformValidityList: A condition in a rule that determines whether or not the rule is enforced by the local computer based on the local computer's platform information.

Origin: This field is the rule origin, as specified in the [FW_RULE_ORIGIN_TYPE](#) enumeration. It MUST be filled on enumerated rules and ignored on input.

wszGPOName: A Unicode string that represents the name of the originating GPO. It MUST be set if the rule origin is Group Policy; otherwise, it MUST be NULL.

Status: The status code of the rule, as specified by the [FW_RULE_STATUS](#) enumeration. This field is filled out when the structure is returned as output. On input, this field MUST be set to `FW_RULE_STATUS_OK`.

wszMMParentRuleId: This field is not used.

Reserved: Not used other than to instruct RPC by using the `FW_OBJECT_CTRL_FLAG_INCLUDE_METADATA` flag that a pointer to a `FW_OBJECT_METADATA` structure is present. It has no semantic meaning to the object itself

pMetaData: A pointer to an `FW_OBJECT_METADATA` structure that contains specific metadata about the current state of the connection security rule.

The following are semantic checks that connection security rules MUST pass:

- The **wSchemaVersion** field MUST NOT be less than 0x000200.
- The **wszRuleId** field MUST NOT contain the pipe '|' character, MUST NOT be NULL, MUST be a string of at least 1 character, and MUST NOT be greater than or equal to 512 characters. [<10>](#)
- The **wszName** field string MUST meet the following criteria:
 - MUST contain at least one character.
 - MUST contain less than 10,000 characters.
 - MUST NOT be NULL.
 - MUST NOT contain the pipe '|' character.
 - MUST NOT equal the string "ALL" (case-insensitive).
- If the **wszDescription** field string is not NULL, it MUST be at least 1 character, MUST NOT be greater than or equal to 10,000 characters, and MUST NOT contain the pipe '|' character.
- If the **wszEmbeddedContext** field string is not NULL, it MUST be at least 1 character, MUST NOT be greater than or equal to 10,000 characters, and MUST NOT contain the pipe '|' character.
- The **dwProfiles** field MUST NOT contain invalid values, and if it is not equal to the ALL profile type, it MUST NOT contain unknown profiles.
- The **wIpProtocol** field MUST NOT be greater than 256.
- If **wIpProtocol** is 6 or 17, the **wPortKeywords** field of **Endpoint1Ports** MUST be 0.

- If **wIpProtocol** is 6 or 17, the **wPortKeywords** field of **Endpoint2Ports** MUST be 0.
- If **wIpProtocol** is neither 6 nor 17, the **Endpoint1Ports** and **Endpoint2Ports** fields MUST be empty.
- If the **Endpoint1** field is not empty, **LocalInterfaceIds** MUST be empty and **dwLocalInterfaceTypes** MUST be 0. If the **Endpoint1** field is empty, **LocalInterfaceIds** MUST NOT be empty and **dwLocalInterfaceTypes** MUST NOT be 0.
- The **Endpoint1** and **Endpoint2** address keywords MUST contain valid address keywords.
- The **Endpoint1** and **Endpoint2** structures MUST NOT contain multicast v4 or v6 addresses.
- The **dwLocalInterfaceTypes** MUST NOT be greater than or equal to **FW_INTERFACE_TYPE_MAX**.
- The **Action** field MUST be a valid action from the **FW_CS_RULE_ACTION** enumeration.
- The **wFlags** field MUST NOT be greater than or equal to **FW_CS_RULE_FLAGS_MAX**.
- If the **Action** field is **FW_CS_RULE_ACTION_DO_NOT_SECURE**, **wszPhase1AuthSet**, **wszPhase2AuthSet**, and **wszPhase2CryptoSet** MUST all be NULL; otherwise, **wszPhase1AuthSet**, **wszPhase2AuthSet**, and **wszPhase2CryptoSet** MUST all be at least 1 character long, MUST NOT be greater than or equal to 1,000 characters, <11> and MUST NOT contain the pipe '|' character.
- A tunnel rule has the **dwRemoteTunnelEndpointV4** (or V6) field as an address or the **dwLocalTunnelEndpointV4** (or V6) as an address. If the rule is a tunnel rule, the **Endpoint1** and **Endpoint2** addresses MUST NOT be empty; the **Action** field MUST be **FW_CS_RULE_ACTION_SECURE**; **wIpProtocol** MUST be ANY (256); **Endpoint1Ports** and **Endpoint2Ports** MUST be empty; and **dwRemoteTunnelEndpointV4** and **dwLocalTunnelEndpointV4** MUST either both be ANY or both be specified. The same applies to v6 tunnel endpoint fields.
- If the rule's **wFlags** field contains the **FW_CS_RULE_FLAGS_DTM** flag, then the rule is also a tunnel rule and the following requirements are relaxed: Either **dwRemoteTunnelEndpointV4** or **dwLocalTunnelEndpointV4**, or both, can now be empty. The same applies to the v6 tunnel endpoint fields. **Endpoint1** or **Endpoint2** or both can now be empty. The action can now also be **FW_CS_RULE_ACTION_DO_NOT_SECURE**.
- Tunnel endpoint addresses MUST NOT be the loopback addresses.
- If the **wFlags** field has the **FW_CS_RULE_FLAGS_OUTBOUND_CLEAR** flag set or the **FW_CS_RULE_FLAGS_TUNNEL_BYPASS_IF_ENCRYPTED** flag set, the rule MUST be a tunnel mode rule.

2.2.54 FW_AUTH_METHOD

This enumeration defines the different authentication methods that are used for authentication.

```
typedef enum _tag_FW_AUTH_METHOD
{
    FW_AUTH_METHOD_INVALID,
    FW_AUTH_METHOD_ANONYMOUS,
    FW_AUTH_METHOD_MACHINE_KERB,
    FW_AUTH_METHOD_MACHINE_SHKEY,
    FW_AUTH_METHOD_MACHINE_NTLM,
}
```



```

FW_AUTH_METHOD_MACHINE_CERT,
FW_AUTH_METHOD_USER_KERB,
FW_AUTH_METHOD_USER_CERT,
FW_AUTH_METHOD_USER_NTLM,
FW_AUTH_METHOD_MAX
} FW_AUTH_METHOD;

```

FW_AUTH_METHOD_INVALID: This value MUST NOT be used. It is defined for simplicity in writing IDL definitions and code. This symbolic constant has a value of zero.

FW_AUTH_METHOD_ANONYMOUS: This method does not require identity to authenticate. It is equal to no authentication. This method can be used on first and second authentications. This symbolic constant has a value of 1.

FW_AUTH_METHOD_MACHINE_KERB: This method authenticates the identity of machines by using [Kerberos Protocol Extensions](#) (as specified in [MS-KILE]). This method MUST be used only on first authentications. This symbolic constant has a value of 2.

FW_AUTH_METHOD_MACHINE_SHKEY: This method uses a previous manually shared key to authenticate machine identities. This method MUST be used only on first authentications. This symbolic constant has a value of 3.

FW_AUTH_METHOD_MACHINE_NTLM: This method authenticates the identity of machines by using the [NTLM Authentication Protocol](#) (as specified in [MS-NLMP]). This method MUST be used only on first authentications. This symbolic constant has a value of 4.

FW_AUTH_METHOD_MACHINE_CERT: This method authenticates the identity of a machine by using machine certificates. This method can be used on both first and second authentications. This symbolic constant has a value of 5.

FW_AUTH_METHOD_USER_KERB: This method authenticates user identities by using the Kerberos Protocol Extensions. This method MUST be used only on second authentications. This symbolic constant has a value of 6.

FW_AUTH_METHOD_USER_CERT: This method authenticates user identities by using user certificates. This method MUST be used only on second authentications. This symbolic constant has a value of 7.

FW_AUTH_METHOD_USER_NTLM: This method authenticates user identities by using the NTLM Authentication Protocol. This method MUST be used only on second authentications. This symbolic constant has a value of 8.

FW_AUTH_METHOD_MAX: This value and greater values are invalid and MUST NOT be used. This symbolic constant is defined for simplicity in writing IDL definitions and code. It has a value of 11.

2.2.55 FW_AUTH_SUITE_FLAGS

This enumeration describes bitmask flags that can be set on authentication proposals.

```

typedef enum _tag_FW_AUTH_SUITE_FLAGS
{
    FW_AUTH_SUITE_FLAGS_NONE = 0x0000,
    FW_AUTH_SUITE_FLAGS_CERT_EXCLUDE_CA_NAME = 0x0001,
    FW_AUTH_SUITE_FLAGS_HEALTH_CERT = 0x0002,
    FW_AUTH_SUITE_FLAGS_PERFORM_CERT_ACCOUNT_MAPPING = 0x0004,

```

```

FW_AUTH_SUITE_FLAGS_CERT_SIGNING_ECDSA256 = 0x0008,
FW_AUTH_SUITE_FLAGS_CERT_SIGNING_ECDSA384 = 0x0010,
FW_AUTH_SUITE_FLAGS_INTERMEDIATE_CA = 0x0020,
FW_AUTH_SUITE_FLAGS_MAX = 0x0040,
FW_AUTH_SUITE_FLAGS_MAX_V2_1 = 0x0020
} FW_AUTH_SUITE_FLAGS;

```

FW_AUTH_SUITE_FLAGS_NONE: This value means that none of the following flags are set. This value is defined for simplicity in writing IDL definitions and code.

FW_AUTH_SUITE_FLAGS_CERT_EXCLUDE_CA_NAME: If this flag is set, certificate authority (CA) names are excluded. This flag **MUST** be set only on first authentications.

FW_AUTH_SUITE_FLAGS_HEALTH_CERT: This flag specifies that the certificate in use is a health certificate. On second authentications, if the authentication method is using a machine certificate, this flag **MUST** be specified. Also on second authentications, if the authentication method is using a user certificate, this flag **MUST NOT** be specified.

FW_AUTH_SUITE_FLAGS_PERFORM_CERT_ACCOUNT_MAPPING: This flag specifies that the certificate that is used maps to an account.

FW_AUTH_SUITE_FLAGS_CERT_SIGNING_ECDSA256: This flag specifies that the default certificate signing algorithm of **RSA** **MUST** be replaced by the Elliptic Curve Digital Signature Algorithm (ECDSA) using curves with a 256-bit prime moduli.

FW_AUTH_SUITE_FLAGS_CERT_SIGNING_ECDSA384: This flag specifies that the default certificate signing algorithm of **RSA** **MUST** be replaced by the Elliptic Curve Digital Signature Algorithm using curves with a 384-bit prime moduli.

FW_AUTH_SUITE_FLAGS_INTERMEDIATE_CA: This flag specifies that the certificate used is not from a root certificate authority but from an intermediate authority in the chain.

FW_AUTH_SUITE_FLAGS_MAX: This value and greater values are invalid and **MUST NOT** be used. It is defined for simplicity in writing IDL definitions and code.

FW_AUTH_SUITE_FLAGS_MAX_V2_1: For schema version 0x0210, this value and greater values are invalid and **MUST NOT** be used. It is defined for simplicity in writing IDL definitions and code.

2.2.56 FW_AUTH_SUITE

This structure describes an IPsec authentication suite. An authentication suite is a proposal of a set of algorithms and parameters that specify the authentication method to be used. It also includes some modifiers and parameters for the authentication method.

```

typedef struct _tag_FW_AUTH_SUITE {
    [range(FW_AUTH_METHOD_INVALID+1, FW_AUTH_METHOD_MAX)]
    FW_AUTH_METHOD Method;
    unsigned short wFlags;
    [switch_type(FW_AUTH_METHOD), switch_is(Method)]
    union {
        [case(FW_AUTH_METHOD_MACHINE_CERT,FW_AUTH_METHOD_USER_CERT)]
        struct {
            [ref, string] wchar_t* wszCAName;
        };
        [case(FW_AUTH_METHOD_MACHINE_SHKEY)]

```

```

        struct {
            [ref, string] wchar_t* wszSHKey;
        };
        [default]          ;
    };
} FW_AUTH_SUITE,
*PFW_AUTH_SUITE;

```

Method: This field is of type [FW_AUTH_METHOD](#). It specifies the authentication method that is suggested by this proposal suite.

wFlags: This flag is a combination of flags from [FW_AUTH_SUITE_FLAGS](#).

wszCAName: A pointer to a Unicode string. This string represents the name of the certificate authority to be used to authenticate when using machine or user certificate methods.

wszSHKey: A pointer to a Unicode string. This string is the previous, manually shared secret that is used to authenticate when using preshared key methods.

If the method is machine certificate or user certificate, the **wszCAName** string MUST NOT be NULL, MUST be at least 1 character long, MUST NOT be greater than or equal to 10,000 characters, MUST NOT contain the pipe(|) character, and MUST be a CERT_X500_NAME_STR string type name encoded with X509_ASN_ENCODING. If the method is SHKEY, the **wszSHKey** string MUST NOT be NULL, MUST be at least 1 character long, MUST NOT be greater than or equal to 10,000 characters, and MUST NOT contain the pipe (|) character.

2.2.57 FW_AUTH_SET

This structure contains a list of [FW_AUTH_SUITE](#) elements that are ordered from highest to lowest preference and are negotiated with remote peers to establish authentication algorithms.

```

typedef struct _tag_FW_AUTH_SET {
    struct _tag_FW_AUTH_SET* pNext;
    unsigned short wSchemaVersion;
    [range(FW_IPSEC_PHASE_INVALID+1, FW_IPSEC_PHASE_MAX-1)]
    FW_IPSEC_PHASE IpSecPhase;
    [string, range(1,255), ref] wchar_t* wszSetId;
    [string, range(1,10001)] wchar_t* wszName;
    [string, range(1,10001)] wchar_t* wszDescription;
    [string, range(1,10001)] wchar_t* wszEmbeddedContext;
    [range(0,1000)] unsigned long dwNumSuites;
    [size_is(dwNumSuites)] PFW_AUTH_SUITE pSuites;
    [range(FW_RULE_ORIGIN_INVALID,FW_RULE_ORIGIN_MAX-1)]
    FW_RULE_ORIGIN_TYPE Origin;
    [string, range(1,10001)] wchar_t* wszGPOName;
    FW_RULE_STATUS Status;
    unsigned long dwAuthSetFlags;
} FW_AUTH_SET,
*PFW_AUTH_SET;

```

pNext: A pointer to the next **FW_AUTH_SET** in the list.

wSchemaVersion: Specifies the version of the set.

IpSecPhase: This field is of type [FW_IPSEC_PHASE](#), and it specifies if this authentication set applies for first or second authentications.

wszSetId: A pointer to a Unicode string that uniquely identifies the set. The default set for this policy object is identified with the "{E5A5D32A-4BCE-4e4d-B07F-4AB1BA7E5FE3}" string for Phase1 and the "{E5A5D32A-4BCE-4e4d-B07F-4AB1BA7E5FE4}" string for Phase2. Default sets are merged across policy stores, and only one is enforced according to predefined merge logic rules.

wszName: A pointer to a Unicode string that provides a friendly name for the set.

wszDescription: A pointer to a Unicode string that provides a friendly description for the set.

wszEmbeddedContext: A pointer to a Unicode string that provides a way for applications to store relevant application-specific context that is related to the set.

dwNumSuites: Specifies the number of authentication suites that the structure contains.

pSuites: A pointer to an array of **FW_AUTH_SUITE** elements. The number of elements is given by **dwNumSuites**.

Origin: This field is the set origin, as specified in the [FW_RULE_ORIGIN_TYPE](#) enumeration. It MUST be filled on enumerated rules and ignored on input.

wszGPOName: A Unicode string that represents the name of the originating GPO. It MUST be set if the origin is Group Policy; otherwise, it MUST be NULL.

Status: A status code of the set, as specified by the [FW_RULE_STATUS](#) enumeration. This field is filled out when the structure is returned as output. On input, this field MUST be set to **FW_RULE_STATUS_OK**.

dwAuthSetFlags: A reserved value and not currently used. It MUST be set to 0.

The following are semantic checks that authentication sets MUST pass:

- The **wSchemaVersion** field MUST NOT be less than 0x000200.
- The **wszSetId** field MUST NOT contain the pipe (|) character, MUST NOT be NULL, MUST be a string of at least 1 character long, and MUST NOT be greater than or equal to 255 characters.
- If the **wszName** field string is not NULL, it MUST be at least 1 character long, MUST NOT be greater than or equal to 10,000 characters, and MUST NOT contain the pipe (|) character.
- If the **wszDescription** field string is not NULL, it MUST be at least 1 character long, MUST NOT be greater than or equal to 10,000 characters, and MUST NOT contain the pipe (|) character.
- If the **wszEmbeddedContext** field string is not NULL, it MUST be at least 1 character long, MUST NOT be greater than or equal to 10,000 characters, and MUST NOT contain the pipe (|) character.
- The **IpSecPhase** field MUST have valid **FW_IPSEC_PHASE** values.
- If **IpSecPhase** is **FW_IPSEC_PHASE_1**:
 - The **wszSetId** field MUST NOT have the default phase 1 authentication set ID as a prefix.
 - The authentication set MUST have at least one authentication suite.

- The **dwNumSuites** field MUST agree with the **pSuites** field.
- The authentication suites methods MUST only be FW_AUTH_METHOD_ANONYMOUS, FW_AUTH_METHOD_MACHINE_KERB, FW_AUTH_METHOD_MACHINE_NTLM, FW_AUTH_METHOD_MACHINE_CERT, or FW_AUTH_METHOD_MACHINE_SHKEY.
- Authentication suites that have a method other than machine certificate MUST have the **wFlags** field of the same suite set to 0.
- If the set schema policy version is 0x200, the **wFlags** field MUST NOT contain the FW_AUTH_SUITE_FLAGS_CERT_SIGNING_ECDSA256 or the FW_AUTH_SUITE_FLAGS_CERT_SIGNING_ECDSA384 flags.
- The **wFlags** field MUST NOT contain both the FW_AUTH_SUITE_FLAGS_CERT_SIGNING_ECDSA256 and the FW_AUTH_SUITE_FLAGS_CERT_SIGNING_ECDSA384 flags.
- All suites that have the FW_AUTH_METHOD_MACHINE_CERT method and a **wFlags** field with the FW_AUTH_SUITE_FLAGS_CERT_SIGNING_ECDSA256 flag set, MUST be contiguous. The same applies for those suites that have the FW_AUTH_SUITE_FLAGS_CERT_SIGNING_ECDSA384 flag set, and those suites that have neither flag set (they default to RSA signing).
- All such contiguous suites that have a specific signing flag (either none, ECDSA256, or ECDSA384) MUST have the same value for the FW_AUTH_SUITE_FLAG_HEALTH_CERT flag. It MUST be set either in all or in none.
- The set MUST NOT have more than one suite that has the anonymous method (FW_AUTH_METHOD_ANONYMOUS), or that has the machine kerb method (FW_AUTH_METHOD_MACHINE_KERB), or that has the machine ntlm method (FW_AUTH_METHOD_MACHINE_NTLM), or that has the machine shkey method (FW_AUTH_METHOD_MACHINE_SHKEY), as defined in section [2.2.54.<12>](#)
- The set MUST NOT have a suite that has an [NTLM Authentication Protocol](#) method (as specified in [MS-NLMP]) and a suite SHKey method.
- If the set has a machine certificate suite that has a **wFlag** that contains the flag FW_AUTH_SUITE_FLAGS_HEALTH_CERT, all machine certificate method suites in the set MUST also have this flag.
- If the **IpSecPhase** is FW_IPSEC_PHASE_2:
 - The **wszSetId** MUST NOT have the default phase 2 authentication set ID as a prefix.
 - The **dwNumSuites** field MUST agree with the **pSuites** field.
 - The authentication suites methods MUST only be either FW_AUTH_METHOD_ANONYMOUS, FW_AUTH_METHOD_USER_KERB, FW_AUTH_METHOD_USER_NTLM, FW_AUTH_METHOD_USER_CERT, or FW_AUTH_METHOD_MACHINE_CERT.
 - The set MUST NOT have a suite that has the anonymous method as the only suite.
 - Suites in the set MUST NOT contain FW_AUTH_SUITE_FLAGS_CERT_EXCLUDE_CA_NAME.
 - Suites that have user certificate methods MUST NOT contain the FW_AUTH_SUITE_FLAGS_HEALTH_CERT flag; however, suites that have machine certificate methods MUST contain it.

- Authentication suites that have a method other than machine certificate or user certificate MUST have the **wFlags** field of the same suite set to 0.
- If the set schema policy version is 0x200, the **wFlags** field MUST NOT contain the FW_AUTH_SUITE_FLAGS_CERT_SIGNING_ECDSA256 or the FW_AUTH_SUITE_FLAGS_CERT_SIGNING_ECDSA384 flags.
- The **wFlags** field MUST NOT contain both the FW_AUTH_SUITE_FLAGS_CERT_SIGNING_ECDSA256 and the FW_AUTH_SUITE_FLAGS_CERT_SIGNING_ECDSA384 flags.
- All suites that have a FW_AUTH_METHOD_MACHINE_CERT method and a **wFlags** field with the FW_AUTH_SUITE_FLAGS_CERT_SIGNING_ECDSA256 flag set, MUST be contiguous. The same applies to those suites that have the FW_AUTH_SUITE_FLAGS_CERT_SIGNING_ECDSA384 flag set and those suites that have neither flag set (they default to RSA signing).
- The set MUST NOT have more than one suite that has the anonymous method (FW_AUTH_METHOD_ANONYMOUS), or that has the user kerb method (FW_AUTH_METHOD_USER_KERB), or that has the user ntlm method (FW_AUTH_METHOD_USER_NTLM), as defined in section [2.2.54.<13>](#)
- A set that contains a suite that has the machine certificate method MUST NOT contain suites that have the user certificate method.
- A set that contains a suite that has the machine certificate method MUST only contain more suites that have machine certificate or anonymous methods.

2.2.58 FW_CRYPTOKEY_EXCHANGE_TYPE

This enumeration is used to identify supported key exchange algorithms.

```
typedef enum _tag_FW_CRYPTOKEY_EXCHANGE_TYPE
{
    FW_CRYPTOKEY_EXCHANGE_NONE = 0,
    FW_CRYPTOKEY_EXCHANGE_DH1,
    FW_CRYPTOKEY_EXCHANGE_DH2,
    FW_CRYPTOKEY_EXCHANGE_ECDH256,
    FW_CRYPTOKEY_EXCHANGE_ECDH384,
    FW_CRYPTOKEY_EXCHANGE_DH2048,
    FW_CRYPTOKEY_EXCHANGE_DH14 = FW_CRYPTOKEY_EXCHANGE_DH2048,
    FW_CRYPTOKEY_EXCHANGE_MAX
} FW_CRYPTOKEY_EXCHANGE_TYPE;
```

FW_CRYPTOKEY_EXCHANGE_NONE: This value means that there are no key exchange algorithms defined. When enumerating SAs, this value MAY be returned. It MUST NOT be used for other cases. This symbolic constant has a value of zero.

FW_CRYPTOKEY_EXCHANGE_DH1: Do key exchange with Diffie-Hellman group 1. This symbolic constant has a value of 1.

FW_CRYPTOKEY_EXCHANGE_DH2: Do key exchange with Diffie-Hellman group 2. This symbolic constant has a value of 2.

FW_CRYPTOKEY_EXCHANGE_ECDH256: Do key exchange with elliptic curve Diffie-Hellman 256. This symbolic constant has a value of 3.

FW_CRYPTO_KEY_EXCHANGE_ECDH384: Do key exchange with elliptic curve Diffie-Hellman 384. This symbolic constant has a value of 4.

FW_CRYPTO_KEY_EXCHANGE_DH2048: Do key exchange with Diffie-Hellman group 2048. This symbolic constant has a value of 5.

FW_CRYPTO_KEY_EXCHANGE_DH14 = FW_CRYPTO_KEY_EXCHANGE_DH2048: Do key exchange with Diffie-Hellman group 14. This group was called Diffie-Hellman group 2048 when it was introduced. The name has been changed to match standard terminology. This symbolic constant has a value of 5.

FW_CRYPTO_KEY_EXCHANGE_MAX: This value and greater values are invalid and MUST NOT be used. It is defined for simplicity in writing IDL definitions and code. This symbolic constant has a value of 6.

2.2.59 FW_CRYPTO_ENCRYPTION_TYPE

This enumeration is used to identify supported encryption algorithms.

```
typedef enum _tag_FW_CRYPTO_ENCRYPTION_TYPE
{
    FW_CRYPTO_ENCRYPTION_NONE,
    FW_CRYPTO_ENCRYPTION_DES,
    FW_CRYPTO_ENCRYPTION_3DES,
    FW_CRYPTO_ENCRYPTION_AES128,
    FW_CRYPTO_ENCRYPTION_AES192,
    FW_CRYPTO_ENCRYPTION_AES256,
    FW_CRYPTO_ENCRYPTION_AES_GCM128,
    FW_CRYPTO_ENCRYPTION_AES_GCM192,
    FW_CRYPTO_ENCRYPTION_AES_GCM256,
    FW_CRYPTO_ENCRYPTION_MAX,
    FW_CRYPTO_ENCRYPTION_MAX_V2_0 = FW_CRYPTO_ENCRYPTION_AES_GCM128
} FW_CRYPTO_ENCRYPTION_TYPE;
```

FW_CRYPTO_ENCRYPTION_NONE: This value MUST be used only when no encryption should be performed. This is a valid value. This symbolic constant has a value of zero.

FW_CRYPTO_ENCRYPTION_DES: Uses the DES algorithm for encryption. This symbolic constant has a value of 1.

FW_CRYPTO_ENCRYPTION_3DES: Uses the 3DES algorithm for encryption. This symbolic constant has a value of 2.

FW_CRYPTO_ENCRYPTION_AES128: Uses the AES algorithm with a 128-bit key size for encryption. This symbolic constant has a value of 3.

FW_CRYPTO_ENCRYPTION_AES192: Uses the AES algorithm with a 192-bit key size for encryption. This symbolic constant has a value of 4.

FW_CRYPTO_ENCRYPTION_AES256: Uses the AES algorithm with a 256-bit key size for encryption. This symbolic constant has a value of 5.

FW_CRYPTO_ENCRYPTION_AES_GCM128: Uses the AESGCM algorithm with a 128-bit key size for encryption. This symbolic constant has a value of 6.

FW_CRYPTO_ENCRYPTION_AES_GCM192: Uses the AESGCM algorithm with a 192-bit key size for encryption. This symbolic constant has a value of 7.

FW_CRYPTO_ENCRYPTION_AES_GCM256: Uses the AESGCM algorithm with a 256-bit key size for encryption. This symbolic constant has a value of 8.

FW_CRYPTO_ENCRYPTION_MAX: This value and greater values are invalid and MUST NOT be used. It is defined for simplicity in writing IDL definitions and code. This symbolic constant has a value of 9.

FW_CRYPTO_ENCRYPTION_MAX_V2_0: For schema version 0x0200, this value and greater values are invalid and MUST NOT be used. This symbolic constant has a value of 6. It is defined for simplicity in writing IDL definitions and describing semantic checks against policy schema versions of 0x0200.

2.2.60 FW_CRYPTO_HASH_TYPE

This enumeration is used to identify the different hashing (integrity protection) algorithms supported.

```
typedef enum _tag_FW_CRYPTO_HASH_TYPE
{
    FW_CRYPTO_HASH_NONE,
    FW_CRYPTO_HASH_MD5,
    FW_CRYPTO_HASH_SHA1,
    FW_CRYPTO_HASH_SHA256,
    FW_CRYPTO_HASH_SHA384,
    FW_CRYPTO_HASH_AES_GMAC128,
    FW_CRYPTO_HASH_AES_GMAC192,
    FW_CRYPTO_HASH_AES_GMAC256,
    FW_CRYPTO_HASH_MAX,
    FW_CRYPTO_HASH_MAX_V2_0 = FW_CRYPTO_HASH_SHA256
} FW_CRYPTO_HASH_TYPE;
```

FW_CRYPTO_HASH_NONE: This value MUST be used only when no hashing should be performed. This is a valid value. This symbolic constant has a value of zero.

FW_CRYPTO_HASH_MD5: Use the MD5 algorithm for hashing (integrity protection). This symbolic constant has a value of 1.

FW_CRYPTO_HASH_SHA1: Use the SHA1 algorithm for hashing (integrity protection). This symbolic constant has a value of 2.

FW_CRYPTO_HASH_SHA256: Use the SHA256 algorithm for hashing (integrity protection). This symbolic constant has a value of 3.

FW_CRYPTO_HASH_SHA384: Use the SHA384 algorithm for hashing (integrity protection). This symbolic constant has a value of 4.

FW_CRYPTO_HASH_AES_GMAC128: Use the AESGMAC128 algorithm for hashing (integrity protection). This symbolic constant has a value of 5.

FW_CRYPTO_HASH_AES_GMAC192: Use the AESGMAC192 algorithm for hashing (integrity protection). This symbolic constant has a value of 6.

FW_CRYPTO_HASH_AES_GMAC256: Use the AESGMAC256 algorithm for hashing (integrity protection). This symbolic constant has a value of 7.

FW_CRYPT_HASH_MAX: This value and greater values are invalid and MUST NOT be used. It is defined for simplicity in writing IDL definitions and code. This symbolic constant has a value of 8.

FW_CRYPT_HASH_MAX_V2_0: For schema version 0x0200, this value and greater values are invalid and MUST NOT be used. This symbolic constant has a value of 3. It is defined for simplicity in writing IDL definitions and describing semantic checks against policy schema versions of 0x0200.

2.2.61 FW_CRYPT_PROTOCOL_TYPE

This enumeration is used to identify the different combinations of supported IPsec enforcement protocols.

```
typedef enum _tag_FW_CRYPT_PROTOCOL_TYPE
{
    FW_CRYPT_PROTOCOL_INVALID,
    FW_CRYPT_PROTOCOL_AH,
    FW_CRYPT_PROTOCOL_ESP,
    FW_CRYPT_PROTOCOL_BOTH,
    FW_CRYPT_PROTOCOL_AUTH_NO_ENCAP,
    FW_CRYPT_PROTOCOL_MAX,
    FW_CRYPT_PROTOCOL_MAX_2_1 = (FW_CRYPT_PROTOCOL_BOTH + 1)
} FW_CRYPT_PROTOCOL_TYPE;
```

FW_CRYPT_PROTOCOL_INVALID: This value MUST NOT be used. It is defined for simplicity in writing IDL definitions and code. This symbolic constant has a value of zero.

FW_CRYPT_PROTOCOL_AH: Uses the **authentication header (AH)** to enforce IPsec. This symbolic constant has a value of 1.

FW_CRYPT_PROTOCOL_ESP: Uses the **ESP** protocol header. This symbolic constant has a value of 2.

FW_CRYPT_PROTOCOL_BOTH: Uses both the AH and ESP protocol headers. This symbolic constant has a value of 3.

FW_CRYPT_PROTOCOL_AUTH_NO_ENCAP: Uses no encapsulation. This sends the first packet twice: once by using an ESP header and again without any header; subsequent packets have no additional headers. This symbolic constant has a value of 4.

FW_CRYPT_PROTOCOL_MAX: This value and greater values invalid and MUST NOT be used. This symbolic constant is defined for simplicity in writing IDL definitions and code. It has a value of 5.

FW_CRYPT_PROTOCOL_MAX_2_1: This value and greater values are invalid and MUST NOT be used. This symbolic constant is defined for simplicity in writing IDL definitions and code.

2.2.62 FW_PHASE1_CRYPT_SUITE

This structure describes an IPsec Phase 1 (or main mode) cryptographic suite. A cryptographic suite is a proposal of a set of algorithms and parameters that specify how different types of enforcement and protection are suggested to be performed.

```
typedef struct _tag_FW_PHASE1_CRYPT_SUITE {
```

```

[range(FW_CRYPTO_KEY_EXCHANGE_NONE, FW_CRYPTO_KEY_EXCHANGE_MAX-1)]
    FW_CRYPTO_KEY_EXCHANGE_TYPE KeyExchange;
[range(FW_CRYPTO_ENCRYPTION_NONE+1, FW_CRYPTO_ENCRYPTION_MAX-1)]
    FW_CRYPTO_ENCRYPTION_TYPE Encryption;
[range(FW_CRYPTO_HASH_NONE+1, FW_CRYPTO_HASH_MAX-1)]
    FW_CRYPTO_HASH_TYPE Hash;
unsigned long dwP1CryptoSuiteFlags;
} FW_PHASE1_CRYPTO_SUITE,
*PFW_PHASE1_CRYPTO_SUITE;

```

KeyExchange: This field is of type [FW_CRYPTO_KEY_EXCHANGE_TYPE](#). It specifies the key exchange algorithm for this suite proposal.

Encryption: This field is of type [FW_CRYPTO_ENCRYPTION_TYPE](#). It specifies the encryption algorithm for this suite proposal.

Hash: This field is of type [FW_CRYPTO_HASH_TYPE](#). It specifies the hash (integrity protection) algorithm for this suite proposal.

dwP1CryptoSuiteFlags: This is a reserved value and is not used. It MUST be set to 0x00000000.

2.2.63 FW_PHASE2_CRYPTO_SUITE

This structure describes an IPsec Phase 2 (or quick mode) cryptographic suite. A cryptographic suite is a proposal of a set of algorithms and parameters that specify how different types of enforcement and protection are suggested to be performed. It also suggests timeouts for which a key is valid and at which re-keying operations should be performed.

```

typedef struct _tag_FW_PHASE2_CRYPTO_SUITE {
    [range(FW_CRYPTO_PROTOCOL_INVALID+1, FW_CRYPTO_PROTOCOL_MAX-1)]
        FW_CRYPTO_PROTOCOL_TYPE Protocol;
    FW_CRYPTO_HASH_TYPE AhHash;
    FW_CRYPTO_HASH_TYPE EspHash;
    FW_CRYPTO_ENCRYPTION_TYPE Encryption;
    unsigned long dwTimeoutMinutes;
    unsigned long dwTimeoutKBytes;
    unsigned long dwP2CryptoSuiteFlags;
} FW_PHASE2_CRYPTO_SUITE,
*PFW_PHASE2_CRYPTO_SUITE;

```

Protocol: This field is of type [FW_CRYPTO_PROTOCOL_TYPE](#), and it specifies the IPsec enforcement protocol combination suggested for this suite.

AhHash: This field is of type [FW_CRYPTO_HASH_TYPE](#). It specifies the hash (integrity protection) algorithm for this suite proposal when using the authentication header protocol.

EspHash: This field is of type [FW_CRYPTO_HASH_TYPE](#). It specifies the hash (integrity protection) algorithm for this suite proposal when using the ESP protocol.

Encryption: This field is of type [FW_CRYPTO_ENCRYPTION_TYPE](#). It specifies the encryption algorithm for this suite proposal.

dwTimeoutMinutes: This is the timeout or lifetime of the key used in this proposal defined in minutes.

dwTimeoutKBytes: This is the timeout or lifetime of the key used in this proposal defined in kilobytes processed with this configuration.

dwP2CryptoSuiteFlags: This field is reserved and is not used. It MUST be set to 0x00000000.

The following are semantic validation checks that Phase 2 cryptographic suites MUST pass:

- The **dwTimeoutMinutes** field MUST be greater than or equal to 5 and less than or equal to 2,879.
- The **dwTimeoutKBytes** field MUST be greater than or equal to 20,480 and less than or equal to 2,147,483,647.
- If the **Protocol** field is FW_CRYPTOPROTOCOL_AH or FW_CRYPTOPROTOCOL_BOTH, the **AhHash** field MUST NOT be equal to FW_CRYPTOPROTOCOL_HASH_NONE.
- If the **Protocol** field is FW_CRYPTOPROTOCOL_BOTH, the **AhHash** field MUST be equal to the **EspHash** field.
- If the **Protocol** field is FW_CRYPTOPROTOCOL_BOTH or FW_CRYPTOPROTOCOL_ESP, **EspHash** MUST NOT be set to FW_CRYPTOPROTOCOL_HASH_NONE or **Encryption** MUST NOT be set to FW_CRYPTOPROTOCOL_ENCRYPTION_NONE, but not both.

2.2.64 FW_PHASE1_CRYPTOPROTOCOL_FLAGS

```
typedef enum _tag_FW_PHASE1_CRYPTOPROTOCOL_FLAGS
{
    FW_PHASE1_CRYPTOPROTOCOL_FLAGS_NONE = 0x00,
    FW_PHASE1_CRYPTOPROTOCOL_FLAGS_DO_NOT_SKIP_DH = 0x01,
    FW_PHASE1_CRYPTOPROTOCOL_FLAGS_MAX = 0x02
} FW_PHASE1_CRYPTOPROTOCOL_FLAGS;
```

FW_PHASE1_CRYPTOPROTOCOL_FLAGS_NONE: This value represents no flag. It is used when none of the behaviors that are represented by the defined flags in the enumeration are intended.

FW_PHASE1_CRYPTOPROTOCOL_FLAGS_DO_NOT_SKIP_DH: This flag ensures that Authenticated IP (AuthIP), as specified in [\[MS-AIPS\]](#), always performs a DH key exchange. (AuthIP can avoid this exchange because the protocol already contains enough key material information to protect the negotiation. Hence, by skipping DH, round trips and the computational cost of DH are avoided.)

FW_PHASE1_CRYPTOPROTOCOL_FLAGS_MAX: This value and greater values are invalid and MUST NOT be used. It is defined for simplicity in writing IDL definitions and code.

2.2.65 FW_PHASE2_CRYPTOPROTOCOL_PFS

This enumeration is used to identify the different perfect forward secrecy (PFS) options supported.

```
typedef enum _tag_FW_PHASE2_CRYPTOPROTOCOL_PFS
{
    FW_PHASE2_CRYPTOPROTOCOL_PFS_INVALID,
    FW_PHASE2_CRYPTOPROTOCOL_PFS_DISABLE,
    FW_PHASE2_CRYPTOPROTOCOL_PFS_PHASE1,
}
```

```

FW_PHASE2_CRYPTOPFS_DH1,
FW_PHASE2_CRYPTOPFS_DH2,
FW_PHASE2_CRYPTOPFS_DH2048,
FW_PHASE2_CRYPTOPFS_ECDH256,
FW_PHASE2_CRYPTOPFS_ECDH384,
FW_PHASE2_CRYPTOPFS_MAX
} FW_PHASE2_CRYPTOPFS;

```

FW_PHASE2_CRYPTOPFS_INVALID: This value MUST NOT be used. It is defined for simplicity in writing IDL definitions and code. This symbolic constant has a value of zero.

FW_PHASE2_CRYPTOPFS_DISABLE: Do not renegotiate; instead, reuse the keying material negotiated in Phase 1 (main mode). This symbolic constant has a value of 1.

FW_PHASE2_CRYPTOPFS_PHASE1: Use Phase 1 key exchange to negotiate a Phase 2 (quick mode) key for every Phase 2 negotiation. This symbolic constant has a value of 2.

FW_PHASE2_CRYPTOPFS_DH1: Use DH1 key exchange to negotiate a Phase 2 (quick mode) key for every Phase 2 negotiation. This symbolic constant has a value of 3.

FW_PHASE2_CRYPTOPFS_DH2: Use DH2 key exchange to negotiate a Phase 2 (quick mode) key for every Phase 2 negotiation. This symbolic constant has a value of 4.

FW_PHASE2_CRYPTOPFS_DH2048: Use DH2048 key exchange to negotiate a Phase 2 (quick mode) key for every Phase 2 negotiation. This symbolic constant has a value of 5.

FW_PHASE2_CRYPTOPFS_ECDH256: Use ECDH256 key exchange to negotiate a Phase 2 (quick mode) key for every Phase 2 negotiation. This symbolic constant has a value of 6.

FW_PHASE2_CRYPTOPFS_ECDH384: Use ECDH384 key exchange to negotiate a Phase 2 (quick mode) key for every Phase 2 negotiation. This symbolic constant has a value of 7.

FW_PHASE2_CRYPTOPFS_MAX: This value and greater values are invalid and MUST NOT be used. It is defined for simplicity in writing IDL definitions and code. This symbolic constant has a value of 8.

2.2.66 FW_CRYPTOPFS_SET

This structure contains a list of cryptographic suite elements that are ordered from highest to lowest preference and are negotiated with remote peers to establish cryptographic protection algorithms.

```

typedef struct _tag_FW_CRYPTOPFS_SET {
    struct _tag_FW_CRYPTOPFS_SET* pNext;
    unsigned short wSchemaVersion;
    [range(FW_IPSEC_PHASE_INVALID+1, FW_IPSEC_PHASE_MAX-1)]
    FW_IPSEC_PHASE IpSecPhase;
    [string, range(1,255), ref] wchar_t* wszSetId;
    [string, range(1,10001)] wchar_t* wszName;
    [string, range(1,10001)] wchar_t* wszDescription;
    [string, range(1,10001)] wchar_t* wszEmbeddedContext;
    [switch_type(FW_IPSEC_PHASE), switch_is(IpSecPhase)]
    union {
        [case(FW_IPSEC_PHASE_1)]
        struct {
            unsigned short wFlags;
            [range(0,1000)] unsigned long dwNumPhase1Suites;
        };
    };
};

```

```

    [size_is(dwNumPhase1Suites)] PFW_PHASE1_CRYPTOSUITE pPhase1Suites;
    unsigned long dwTimeoutMinutes;
    unsigned long dwTimeoutSessions;
};
[case(FW_IPSEC_PHASE_2)]
    struct {
        FW_PHASE2_CRYPTOPFS Pfs;
        [range(0,1000)] unsigned long dwNumPhase2Suites;
        [size_is(dwNumPhase2Suites)] PFW_PHASE2_CRYPTOSUITE pPhase2Suites;
    };
};
[range(FW_RULE_ORIGIN_INVALID,FW_RULE_ORIGIN_MAX-1)]
    FW_RULE_ORIGIN_TYPE Origin;
[string, range(1,10001)] wchar_t* wszGPOName;
    FW_RULE_STATUS Status;
    unsigned long dwCryptoSetFlags;
} FW_CRYPTOSSET,
*PFW_CRYPTOSSET;

```

pNext: A pointer to the next **FW_CRYPTOSSET** in the list.

wSchemaVersion: Specifies the version of the set.

IpSecPhase: This field is of type [FW_IPSEC_PHASE](#), and it specifies if this cryptographic set applies for Phase1 (main mode) or Phase2 (quick mode).

wszSetId: A pointer to a Unicode string that uniquely identifies the set. The default set for this policy object is identified with the "{E5A5D32A-4BCE-4e4d-B07F-4AB1BA7E5FE1}" string for Phase1 and with the "{E5A5D32A-4BCE-4e4d-B07F-4AB1BA7E5FE2}" string for Phase2. Default sets are merged across policy stores, and only one is enforced according to predefined merge logic rules.

wszName: A pointer to a Unicode string that provides a friendly name for the set.

wszDescription: A pointer to a Unicode string that provides a friendly description for the set.

wszEmbeddedContext: A pointer to a Unicode string that provides a way for applications to store relevant application-specific context related to the set.

wFlags: This field is a combination of the [FW_PHASE1_CRYPTOSSET_FLAGS](#) enumeration bit flags.

dwNumPhase1Suites: Specifies the number of Phase1 suites that the structure contains.

pPhase1Suites: A pointer to an array of **dwNumPhase1Suites** contiguous [FW_PHASE1_CRYPTOSUITE](#) elements.

dwTimeoutMinutes: This value is a lifetime in minutes before a Phase1 established key is renegotiated.

dwTimeoutSessions: This value is the number of sessions before a Phase1 established key is renegotiated.

Pfs: This field MUST contain a valid value of those in the [FW_PHASE2_CRYPTOPFS](#) enumeration. It describes the perfect forward secrecy used for quick mode cryptographic operations.

dwNumPhase2Suites: Specifies the number of Phase2 suites that the structure contains.

pPhase2Suites: A pointer to an array of [FW_PHASE2_CRYPTO_SUITE](#) elements. The number of elements is given by **dwNumPhase2Suites**.

Origin: This field is the set origin, as specified in the [FW_RULE_ORIGIN_TYPE](#) enumeration. It MUST be filled on enumerated rules and ignored on input.

wszGPOName: A pointer to a Unicode string that represents the name of the originating GPO. It MUST be set if the origin is Group Policy; otherwise, it MUST be NULL.

Status: The status code of the set, as specified by the [FW_RULE_STATUS](#) enumeration. This field is filled out when the structure is returned as output. On input, this field MUST be set to `FW_RULE_STATUS_OK`.

dwCryptoSetFlags: A reserved value, and not currently used. It MUST be set to 0.

The following are semantic checks that cryptographic sets MUST pass:

- The **wSchemaVersion** field MUST NOT be less than 0x000200.
- The **wszSetId** field MUST NOT contain the pipe (|) character, MUST NOT be NULL, MUST be a string at least 1 character long, and MUST NOT be greater than or equal to 255 characters.
- If the **wszName** field string is not NULL, it MUST be at least 1 character long, MUST NOT be greater than or equal to 10,000 characters, and MUST NOT contain the pipe (|) character.
- If the **wszDescription** field string is not NULL, it MUST be at least 1 character long, MUST NOT be greater than or equal to 10,000 characters, and MUST NOT contain the pipe (|) character.
- If the **wszEmbeddedContext** field string is not NULL, it MUST be at least 1 character long, MUST NOT be greater than or equal to 10,000 characters, and MUST NOT contain the pipe (|) character.
- The **IpSecPhase** field MUST have valid **FW_IPSEC_PHASE** values.
- If the **IpSecPhase** field is `FW_IPSEC_PHASE_1`:
 - The **wszSetId** field MUST be equal to the default Phase1 cryptographic set ID. (There is only one Phase1 cryptographic set allowed per store.)
 - The **wFlags** field of the set MUST NOT be greater than or equal to `FW_PHASE1_CRYPT_FLAGS_MAX`.
 - The **dwTimeoutMinutes** field of the set MUST be greater than or equal to 1, and MUST be less than or equal to 2,879.
 - The **dwTimeoutSessions** field of the set MUST be less than or equal to 2,147,483,647.
 - The cryptographic set MUST have at least one Phase1 cryptographic suite.
 - The **pPhase1Suites** array MUST contain exactly **dwNumPhase1Suites** entries.
 - All cryptographic suites within the set MUST have the same value in the **KeyExchange** field and MUST have valid values.
 - All Phase1 suites MUST NOT have a **KeyExchange** field with the `FW_CRYPT_ENCRYPTION_INVALID` value and MUST have valid values.

- If the set has a schema policy version of 0x0200, all Phase1 suites MUST NOT have an **Encryption** field with values greater than or equal to FW_CRYPT0_ENCRYPTION_MAX_V2_0.
- All Phase1 suites MUST NOT have an **Encryption** field with the FW_CRYPT0_ENCRYPTION_NONE value and MUST have valid values less than FW_CRYPT0_ENCRYPTION_MAX_V2_0.
- If the set has a schema policy version of 0x0200, all Phase1 suites MUST NOT have a **Hash** field that has values greater than or equal to FW_CRYPT0_HASH_MAX_V2_0.
- All Phase1 suites MUST NOT have a **Hash** field that has the FW_CRYPT0_HASH_NONE value and MUST have either MD5 (FW_CRYPT0_HASH_MD5) or SHA (FW_CRYPT0_HASH_SHA1, FW_CRYPT0_HASH_SHA256, FW_CRYPT0_HASH_SHA384) valid values.
- If the **IpSecPhase** field is FW_IPSEC_PHASE_2:
 - The **wszSetId** field MUST NOT have the default Phase2 cryptographic set ID as a prefix.
 - The cryptographic set MUST have at least one Phase2 cryptographic suite.
 - The **pPhase2Suites** array MUST contain exactly **dwNumPhase2Suites** entries.
 - The **Pfs** field MUST NOT be FW_PHASE2_CRYPT0_PFS_INVALID and MUST have valid values.
 - If the set has a schema policy version of 0x0200, all Phase2 cryptographic suites MUST NOT have an **AhHash** field or **EspHash** field with values greater than or equal to FW_CRYPT0_HASH_MAX_V2_0.
 - If the set has a schema policy version of 0x0200, all Phase2 suites MUST NOT have an **Encryption** field with values greater than or equal to FW_CRYPT0_ENCRYPTION_MAX_V2_0.
 - All Phase2 suites within the set MUST NOT have a **dwTimeoutMinutes** field less than FW_MIN_CRYPT0_PHASE2_TIMEOUT_MINUTES (5) or greater than FW_MAX_CRYPT0_PHASE2_TIMEOUT_MINUTES (48 * 60 -1).
 - All Phase2 suites within the set MUST NOT have a **dwTimeoutKBytes** field of less than FW_MIN_CRYPT0_PHASE2_TIMEOUT_KBYTES (20480) or greater than FW_MAX_CRYPT0_PHASE2_TIMEOUT_KBYTES (2147483647).
 - All the Phase2 suites within the set MUST NOT have a **Protocol** field with FW_CRYPT0_PROTOCOL_INVALID and MUST have valid values.
 - For all suites that have the **Protocol** field equal to FW_CRYPT0_PROTOCOL_AH or to FW_CRYPT0_PROTOCOL_BOTH:
 - All suites MUST NOT have an **AhHash** field with the FW_CRYPT0_HASH_NONE value, and MUST have valid values not equal to FW_CRYPT0_HASH_SHA384.
 - For all suites that have the **Protocol** field equal to FW_CRYPT0_PROTOCOL_BOTH:
 - All suites MUST have the **AhHash** field equal to the **EspHash** field.
 - For all suites that have the **Protocol** field equal to FW_CRYPT0_PROTOCOL_ESP:
 - All suites MUST have an **EspHash** field with valid values, including FW_CRYPT0_HASH_NONE. The **EspHash** field MUST NOT equal FW_CRYPT0_HASH_SHA384.

- All suites MUST have an **Encryption** field with valid values, including FW_CRYPTO_ENCRYPTION_NONE.
- All suites MUST not have both the **EspHash** field equal to FW_CRYPTO_HASH_NONE and the **Encryption** field equal to FW_CRYPTO_ENCRYPTION_NONE.
- All suites that have the **Encryption** field equal to FW_CRYPTO_ENCRYPTION_AES_GCM128, 192, or 256 MUST also have a corresponding FW_CRYPTO_HASH_AES_GMAC128, 192, or 256 value on the **EspHash** field. An AES GCM encryption algorithm corresponds to an AES GMAC hash algorithm if both use the same bit size.

2.2.67 FW_BYTE_BLOB

This structure contains a memory section. The format of the memory is defined by the context where it is used; for example, see the **SubjectName** field of the [FW_CERT_INFO](#) structure.

```
typedef struct _tag_FW_BYTE_BLOB {
    [range(0,10000)] unsigned long dwSize;
    [size_is(dwSize)] unsigned char* Blob;
} FW_BYTE_BLOB,
*PFW_BYTE_BLOB;
```

dwSize: This field specifies the size in octets of the **Blob** field.

Blob: A pointer to an array of **dwSize** octets.

2.2.68 FW_COOKIE_PAIR

This structure holds random numbers generated out of IPsec negotiations.

```
typedef struct _tag_FW_COOKIE_PAIR {
    unsigned __int64 Initiator;
    unsigned __int64 Responder;
} FW_COOKIE_PAIR,
*PFW_COOKIE_PAIR;
```

Initiator: A random number that maps to the negotiated state that is a security association of the machine that initiated communication and, hence, initiated IKE/AuthIP (for more information, see [RFC2409](#)) as specified in [MS-IKEE](#) and [MS-AIPS](#) traffic.

Responder: A random number that maps to the negotiated state that is a security association of the machine that responded to the communication and, hence, responded to the IKE/AuthIP traffic.

2.2.69 FW_PHASE1_KEY_MODULE_TYPE

This enumeration identifies the different IPsec Key Exchange negotiation protocols that can be used.

```
typedef enum _tag_FW_PHASE1_KEY_MODULE_TYPE
{
    FW_PHASE1_KEY_MODULE_INVALID,
```



```

FW_PHASE1_KEY_MODULE_IKE,
FW_PHASE1_KEY_MODULE_AUTH_IP,
FW_PHASE1_KEY_MODULE_MAX
} FW_PHASE1_KEY_MODULE_TYPE;

```

FW_PHASE1_KEY_MODULE_INVALID: This value MUST NOT be used. It is defined for simplicity in writing IDL definitions and code.

FW_PHASE1_KEY_MODULE_IKE: The keying protocol was IKE.

FW_PHASE1_KEY_MODULE_AUTH_IP: The keying protocol was AuthIP.

FW_PHASE1_KEY_MODULE_MAX: This value and greater values are invalid and MUST NOT be used. It is defined for simplicity in writing IDL definitions and code.

2.2.70 FW_CERT_INFO

This structure represents information on the certificate used in the certificate-based authentication mechanisms.

```

typedef struct _tag_FW_CERT_INFO {
    FW_BYTE_BLOB SubjectName;
    [range(FW_AUTH_SUITE_FLAGS_NONE, FW_AUTH_SUITE_FLAGS_MAX-1)]
    unsigned long dwCertFlags;
} FW_CERT_INFO,
*PFW_CERT_INFO;

```

SubjectName: The subject name of the certificate represented as a [FW_BYTE_BLOB](#) type. This BLOB is an ASN.1-encoded sequence of RDN attributes.

dwCertFlags: This field can be a combination of bit flags from [FW_AUTH_SUITE_FLAGS](#). This field MUST use only health certificate or certificate to account mapping flags, which represent certificate characteristics.

2.2.71 FW_AUTH_INFO

This structure contains information on the local and remote hosts that resulted from the authentication methods performed between them.

```

typedef struct _tag_FW_AUTH_INFO {
    [range(FW_AUTH_METHOD_INVALID + 1, FW_AUTH_METHOD_MAX)]
    FW_AUTH_METHOD AuthMethod;
    [switch_type(FW_AUTH_METHOD), switch_is(AuthMethod)]
    union ;
    unsigned long dwAuthInfoFlags;
} FW_AUTH_INFO,
*PFW_AUTH_INFO;

```

AuthMethod: This field contains the authentication method used to establish the identities of the endpoints and is stored in the security association. The field can take valid values from the [FW_AUTH_METHOD](#) enumeration.

dwAuthInfoFlags: Reserved value and not currently used. It MUST be set to 0.

2.2.72 FW_ENDPOINTS

This structure represents the two endpoints, source and destination, that participate in IP communication.

```
typedef struct _tag_FW_ENDPOINTS {
    [range(FW_IP_VERSION_INVALID+1,FW_IP_VERSION_MAX-1)]
    FW_IP_VERSION IpVersion;
    unsigned long dwSourceV4Address;
    unsigned long dwDestinationV4Address;
    unsigned char SourceV6Address[16];
    unsigned char DestinationV6Address[16];
} FW_ENDPOINTS,
*PFW_ENDPOINTS;
```

IpVersion: This field specifies the Internet Protocol version used. This field MUST contain a valid value from the [FW_IP_VERSION](#) enumeration.

dwSourceV4Address: This field is the IPv4 address of the source endpoint.

dwDestinationV4Address: This field is the IPv4 address of the destination endpoint.

SourceV6Address: This field is a 16-octet array that represents the IPv6 address of the source endpoint.

DestinationV6Address: This field is a 16-octet array that represents the IPv6 address of the destination endpoint.

The v4 versions or the v6 versions of the fields are used depending on the **IpVersion** field value.

2.2.73 FW_PHASE1_SA_DETAILS

This structure represents a security association that is established after the main mode negotiations take place; it contains the selected algorithms to enforce IPsec and the methods and results of the authentication process.

```
typedef struct _tag_FW_PHASE1_SA_DETAILS {
    unsigned __int64 SaId;
    [range(FW_PHASE1_KEY_MODULE_INVALID+1,FW_PHASE1_KEY_MODULE_MAX-1)]
    FW_PHASE1_KEY_MODULE_TYPE KeyModuleType;
    FW_ENDPOINTS Endpoints;
    FW_PHASE1_CRYPTO_SUITE SelectedProposal;
    unsigned long dwProposalLifetimeKBytes;
    unsigned long dwProposalLifetimeMinutes;
    unsigned long dwProposalMaxNumPhase2;
    FW_COOKIE_PAIR CookiePair;
    PFW_AUTH_INFO pFirstAuth;
    PFW_AUTH_INFO pSecondAuth;
    unsigned long dwP1SaFlags;
} FW_PHASE1_SA_DETAILS,
*PFW_PHASE1_SA_DETAILS;
```

SaId: A 64-bit integer that uniquely identifies the security association.

KeyModuleType: The keying protocol used, IKE or AuthIP. The field MUST contain only a value from the [FW_PHASE1_KEY_MODULE_TYPE](#) enumeration.

Endpoints: This field contains IP address information of the two endpoints that established this security association. An address of zero means the security association applies to any endpoint.

SelectedProposal: This is the Phase1 cryptographic suite that was selected by the negotiation of the keying protocol.

dwProposalLifetimeKBytes: Currently not supported.

dwProposalLifetimeMinutes: This field specifies the lifetime in minutes of this security association before a rekey must happen.

dwProposalMaxNumPhase2: This field specifies the number of Phase2 (quick mode) negotiations (rekeys) that can happen before this security association must be renegotiated.

CookiePair: This value is used for diagnostics.

pFirstAuth: A pointer to an [FW_AUTH_INFO](#) structure that contains the information that resulted from the method negotiated and used for first authentication. This pointer MUST NOT be null.

pSecondAuth: A pointer to an [FW_AUTH_INFO](#) structure that contains the information that resulted from the method negotiated and used for second authentication. If the field is NULL, the second authentication was not performed.

dwP1SaFlags: Reserved value and not currently used. It MUST be set to 0.

2.2.74 FW_PHASE2_TRAFFIC_TYPE

This enumeration identifies the two types of traffic enforcement modes that IPsec supports. It is defined in the IDL for future use.

```
typedef enum _tag_FW_PHASE2_TRAFFIC_TYPE
{
    FW_PHASE2_TRAFFIC_TYPE_INVALID,
    FW_PHASE2_TRAFFIC_TYPE_TRANSPORT,
    FW_PHASE2_TRAFFIC_TYPE_TUNNEL,
    FW_PHASE2_TRAFFIC_TYPE_MAX
} FW_PHASE2_TRAFFIC_TYPE;
```

FW_PHASE2_TRAFFIC_TYPE_INVALID: This value MUST NOT be used. It is defined for simplicity in writing IDL definitions and code. This symbolic constant has a value of zero.

FW_PHASE2_TRAFFIC_TYPE_TRANSPORT: This value represents IPsec transport mode, which happens directly between two endpoints. This symbolic constant has a value of 1.

FW_PHASE2_TRAFFIC_TYPE_TUNNEL: This value represents IPsec tunnel mode, which uses two other endpoints to tunnel through them when the original endpoints communicate. This symbolic constant has a value of 2.

FW_PHASE2_TRAFFIC_TYPE_MAX: This value and greater values are invalid and MUST NOT be used. It is defined for simplicity in writing IDL definitions and code. This symbolic constant has a value of 3.

2.2.75 FW_PHASE2_SA_DETAILS

This structure represents a security association that is established after the quick mode negotiations take place; it contains the selected algorithms to enforce IPsec.

```
typedef struct _tag_FW_PHASE2_SA_DETAILS {
    unsigned __int64 SaId;
    [range(FW_DIR_INVALID+1,FW_DIR_MAX-1)]
    FW_DIRECTION Direction;
    FW_ENDPOINTS Endpoints;
    unsigned short wLocalPort;
    unsigned short wRemotePort;
    unsigned short wIpProtocol;
    FW_PHASE2_CRYPTO_SUITE SelectedProposal;
    FW_PHASE2_CRYPTO_PFS Pfs;
    GUID TransportFilterId;
    unsigned long dwP2SaFlags;
} FW_PHASE2_SA_DETAILS,
*PFW_PHASE2_SA_DETAILS;
```

SaId: A 64-bit integer number that uniquely identifies the security association.

Direction: This field specifies the direction of the traffic this security association is securing.

Endpoints: This field contains IP address information of the two endpoints that established this security association. An address of zero means the security association applies to any endpoint.

wLocalPort: This field specifies the port of the local endpoint that is used in the traffic secured by this security association. A value of 0 specifies any port.

wRemotePort: This field specifies the port of the remote endpoint that is used in the traffic secured by this security association. A value of 0 specifies any port.

wIpProtocol: This field specifies the protocol of the traffic secured by this security association. If the value is within the range 0 to 255, the value describes a protocol as in IETF IANA numbers (for more information, see [\[IANA-PROTO-NUM\]](#)). If the value is 256, the rule matches ANY protocol.

SelectedProposal: This field contains the Phase2 cryptographic suite selected by the negotiation that is used by this security association to enforce IPsec.

Pfs: This field specifies the perfect forward secrecy used by this security association.

TransportFilterId: This GUID uniquely identifies the enforcement state object used to generate this security association. This GUID is in no way an encoding of the enforcement state object it identifies. Enforcement state objects are implementation-specific objects that can be added, deleted, or modified and do the job of enforcing actions on the traffic in the network stack and, in this case, do the job of generating the specific security association described in this section. [<14>](#) This GUID currently has no potential use over the wire, as no other API can

receive this identifier. However, other implementations are free to expose such an API for whatever purpose is deemed appropriate.

dwP2SaFlags: Reserved value and not currently used. It MUST be set to 0.

2.2.76 FW_PROFILE_CONFIG_VALUE

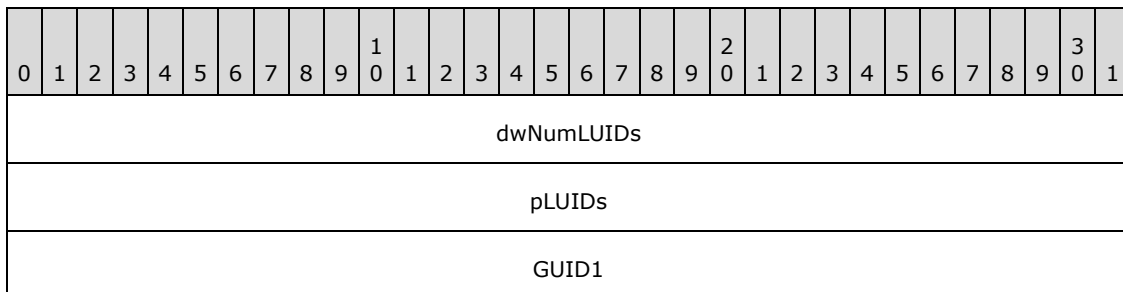
This union defines the value stored by each of the different policy configuration values identified by the enumeration [FW_PROFILE_CONFIG](#). This data type is used to pass different types of values across the same structure on function calls.

```
typedef
[switch_type(FW_PROFILE_CONFIG)]
union _FW_PROFILE_CONFIG_VALUE {
[case(FW_PROFILE_CONFIG_LOG_FILE_PATH)]
    [string, range(1,10001)] wchar_t* wszStr;
[case(FW_PROFILE_CONFIG_DISABLED_INTERFACES)]
    PFW_INTERFACE_LUIDS pDisabledInterfaces;
[case(FW_PROFILE_CONFIG_ENABLE_FW,
    FW_PROFILE_CONFIG_DISABLE_STEALTH_MODE,
    FW_PROFILE_CONFIG_SHIELDED,
    FW_PROFILE_CONFIG_DISABLE_UNICAST_RESPONSES_TO_MULTICAST_BROADCAST,
    FW_PROFILE_CONFIG_LOG_DROPPED_PACKETS,
    FW_PROFILE_CONFIG_LOG_SUCCESS_CONNECTIONS,
    FW_PROFILE_CONFIG_LOG_IGNORED_RULES,
    FW_PROFILE_CONFIG_LOG_MAX_FILE_SIZE,
    FW_PROFILE_CONFIG_DISABLE_INBOUND_NOTIFICATIONS,
    FW_PROFILE_CONFIG_AUTH_APPS_ALLOW_USER_PREF_MERGE,
    FW_PROFILE_CONFIG_GLOBAL_PORTS_ALLOW_USER_PREF_MERGE,
    FW_PROFILE_CONFIG_ALLOW_LOCAL_POLICY_MERGE,
    FW_PROFILE_CONFIG_ALLOW_LOCAL_IPSEC_POLICY_MERGE,
    FW_PROFILE_CONFIG_DEFAULT_OUTBOUND_ACTION,
    FW_PROFILE_CONFIG_DEFAULT_INBOUND_ACTION)]
    unsigned long* pdwVal;
} FW_PROFILE_CONFIG_VALUE,
*PFW_PROFILE_CONFIG_VALUE;
```

wszStr: This field contains a pointer to a Unicode string. It is used when the data type of the configuration value is a string.

pDisabledInterfaces: This field contains a pointer to an [FW_INTERFACE_LUIDS](#) data type, which holds a list of GUIDs. This field is custom marshaled, so it is passed as a plain buffer. The following diagrams show how the structures are marshaled.

On 32-bit servers:



...
...
...

On 64-bit servers:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
dwNumLUIDs																															
Padding for 64-bit alignment																															
pLUIDs																															
...																															
GUID1																															
...																															
...																															
...																															

pdwVal: This field contains a pointer to an **unsigned long**. It is used when the data type of the configuration value is an **unsigned long**.

2.2.77 FW_MM_RULE

This structure is used to represent a main mode rule.

```
typedef struct _tag_FW_MM_RULE {
    struct _tag_FW_MM_RULE* pNext;
    unsigned SHORT wSchemaVersion;
    [string, range(1,512), ref] wchar_t* wszRuleId;
    [string, range(1,10001)] wchar_t* wszName;
    [string, range(1,10001)] wchar_t* wszDescription;
    unsigned LONG dwProfiles;
    FW_ADDRESSES Endpoint1;
    FW_ADDRESSES Endpoint2;
    [string, range(1,255)] wchar_t* wszPhase1AuthSet;
    [string, range(1,255)] wchar_t* wszPhase1CryptoSet;
    unsigned SHORT wFlags;
    [string, range(1,10001)] wchar_t wszEmbeddedContext;
    FW_OS_PLATFORM_LIST PlatformValidityList;
    [range(FW_RULE_ORIGIN_INVALID, FW_RULE_ORIGIN_MAX-1)]
        FW_RULE_ORIGIN_TYPE Origin;
}
```

```

[string, range(1,10001)] wchar_t wszGPOName;
FW_RULE_STATUS Status;
signed LONG Reserved;
[size_is((Reserved & FW_OBJECT_CTRL_FLAG_INCLUDE_METADATA) ? 1 : 0)]
    FW_OBJECT_METADATA pMetaData;
} FW_MM_RULE,
*PFW_MM_RULE;

```

pNext: A pointer to the next **FW_MM_RULE** in the list.

wSchemaVersion: Specifies the version of the rule.

wszRuleId: A pointer to a Unicode string that uniquely identifies the rule.

wszName: A pointer to a Unicode string that provides a friendly name for the rule.

wszDescription: A pointer to a Unicode string that provides a friendly description for the rule.

dwProfiles: A bitmask of the [FW_PROFILE_TYPE](#) flags. It is a condition that matches traffic on the specified profiles.

EndPoint1: A condition that specifies the addresses of the first host of the traffic that the rule matches. An empty EndPoint1 structure means this condition is not applied (no match).

EndPoint2: A condition that specifies the addresses of the second host of the traffic that the rule matches. An empty EndPoint2 structure means this condition is not applied (no match).

wszPhase1AuthSet: A Unicode string that represents the set identifier of a Phase1 authentication sets policy objects.

wFlags: Bit flags from [FW_CS_RULE_FLAGS](#).

wszEmbeddedContext: A pointer to a Unicode string that specifies a group name for this rule. Other components in the system use this string to enable or disable a group of rules by verifying that all rules have the same group name.

PlatformValidityList: A condition that specifies the platforms that send or receive the traffic that the rule matches.

Origin: This field is the rule origin, as specified in the [FW_RULE_ORIGIN_TYPE](#) enumeration. It MUST be filled on enumerated rules and ignored on input.

wszGPOName: A Unicode string that represents the name of the originating GPO. It MUST be set if the rule origin is Group Policy; otherwise, it MUST be null.

Status: The status code of the rule, as specified by the [FW_RULE_STATUS](#) enumeration. This field is filled out when the structure is returned as output. On input, this field MUST be set to FW_RULE_STATUS_OK.

Reserved: This member is not used, other than to instruct RPC, by using the FW_OBJECT_CTRL_FLAG_INCLUDE_METADATA flag, that a pointer to an [FW_OBJECT_METADATA](#) structure is present. It has no semantic meaning to the object itself.

pMetaData: A pointer to an **FW_OBJECT_METADATA** structure that contains specific metadata about the current state of the connection security rule.

2.2.78 FW_CONN_HANDLE

This type contains an RPC binding handle, as specified in [\[C706\]](#) section 2, to an RPC interface that implements the Firewall and Advanced Security Protocol.

This type is declared as follows:

```
typedef handle_t FW_CONN_HANDLE;
```

2.2.79 FW_MATCH_KEY

This enumeration describes the keys that a query is allowed to match.

```
typedef enum _tag_FW_MATCH_KEY
{
    FW_MATCH_KEY_PROFILE,
    FW_MATCH_KEY_STATUS,
    FW_MATCH_KEY_OBJECTID,
    FW_MATCH_KEY_FILTERID,
    FW_MATCH_KEY_APP_PATH,
    FW_MATCH_KEY_PROTOCOL,
    FW_MATCH_KEY_LOCAL_PORT,
    FW_MATCH_KEY_REMOTE_PORT,
    FW_MATCH_KEY_GROUP,
    FW_MATCH_KEY_SVC_NAME,
    FW_MATCH_KEY_DIRECTION,
    FW_MATCH_KEY_MAX
} FW_MATCH_KEY;
```

FW_MATCH_KEY_PROFILE: This key matches the profile conditions of the queried object. This symbolic constant has a value of zero.

FW_MATCH_KEY_STATUS: This key matches the status conditions of the queried object. This symbolic constant has a value of 1.

FW_MATCH_KEY_OBJECTID: This key matches the object ID (rule ID or set ID) of the queried object. This symbolic constant has a value of 2.

FW_MATCH_KEY_FILTERID: This value is not used on the wire. This symbolic constant has a value of 3.

FW_MATCH_KEY_APP_PATH: This key matches the application condition of the queried object. This symbolic constant has a value of 4.

FW_MATCH_KEY_PROTOCOL: This key matches the protocol condition of the queried object. This symbolic constant has a value of 5.

FW_MATCH_KEY_LOCAL_PORT: This key matches the TCP or UDP local port condition of the queried object. This symbolic constant has a value of 6.

FW_MATCH_KEY_REMOTE_PORT: This key matches the TCP or UDP remote port condition of the queried object. This symbolic constant has a value of 7.

FW_MATCH_KEY_GROUP: This key matches the group name (the Embedded context field) of the queried object. This symbolic constant has a value of 8.

FW_MATCH_KEY_SVC_NAME: This key matches the service name condition of the queried object. This symbolic constant has a value of 9.

FW_MATCH_KEY_DIRECTION: This key matches the direction condition of the queried object. This symbolic constant has a value of 10.

FW_MATCH_KEY_MAX: This value and values that exceed this value are not valid and MUST NOT be used. This symbolic constant is defined for simplicity in writing IDL definitions and code. It has a value of 11.

2.2.80 FW_DATA_TYPE

This enumeration describes the data types that this protocol uses in generic structures. It is currently used only in section [2.2.81](#).

```
typedef enum _tag_FW_DATA_TYPE
{
    FW_DATA_TYPE_EMPTY,
    FW_DATA_TYPE_UINT8,
    FW_DATA_TYPE_UINT16,
    FW_DATA_TYPE_UINT32,
    FW_DATA_TYPE_UINT64,
    FW_DATA_TYPE_UNICODE_STRING
} FW_DATA_TYPE;
```

FW_DATA_TYPE_EMPTY: The value SHOULD be empty and not used. This symbolic constant has a value of zero.

FW_DATA_TYPE_UINT8: This data type is a [UINT8](#), which is an 8-bit unsigned integer. This symbolic constant has a value of 1.

FW_DATA_TYPE_UINT16: This data type is a [UINT16](#), which is a 16-bit unsigned integer. This symbolic constant has a value of 2.

FW_DATA_TYPE_UINT32: This data type is a [UINT32](#), which is a 32-bit unsigned integer. This symbolic constant has a value of 3.

FW_DATA_TYPE_UINT64: This data type is a [UINT64](#), which is a 64-bit unsigned integer. This symbolic constant has a value of 4.

FW_DATA_TYPE_UNICODE_STRING: This data type is a Unicode string. This symbolic constant has a value of 5.

2.2.81 FW_MATCH_VALUE

This structure is used to generically store different data types.

```
typedef struct _tag_FW_MATCH_VALUE {
    FW_DATA_TYPE type;
    [switch_type(FW_DATA_TYPE), switch_is(type)]
    union {
        [case(FW_DATA_TYPE_UINT8)]
        unsigned CHAR uInt8;
    };
};
```

```

[case (FW_DATA_TYPE_UINT16)]
    unsigned SHORT uInt16;
[case (FW_DATA_TYPE_UINT32)]
    unsigned LONG uInt32;
[case (FW_DATA_TYPE_UINT64)]
    unsigned __int64 uInt64;
[case (FW_DATA_TYPE_UNICODE_STRING)]
    struct {
        [string, range(1,10001)] wchar_t* wszString;
    };
[case (FW_DATA_TYPE_EMPTY)]
    ;
};
} FW_MATCH_VALUE;

```

type: This field identifies the data type that is stored in the structure.

uInt8: This field contains an 8-bit unsigned integer.

uInt16: This field contains a 16-bit unsigned integer.

uInt32: This field contains a 32-bit unsigned integer.

uInt64: This field contains a 64-bit unsigned integer.

wszString: This field contains a pointer to a Unicode string.

2.2.82 FW_MATCH_TYPE

This enumeration specifies how a match key is matched against an object.

```

typedef enum _tag_FW_MATCH_TYPE
{
    FW_MATCH_TYPE_TRAFFIC_MATCH,
    FW_MATCH_TYPE_EQUAL,
    FW_MATCH_TYPE_MAX
} FW_MATCH_TYPE;

```

FW_MATCH_TYPE_TRAFFIC_MATCH: The match operation evaluates to TRUE for all objects that match the network traffic that is represented by the value matched against. This symbolic constant has a value of 0.

FW_MATCH_TYPE_EQUAL: The match operation evaluates to TRUE for all objects that have a value equal to the one matched against. This symbolic constant has a value of 1.

FW_MATCH_TYPE_MAX: This value and values that exceed this value are not valid and MUST NOT be used. This symbolic constant is defined for simplicity in writing IDL definitions and code. It has a value of 2.

2.2.83 FW_QUERY_CONDITION

This structure specifies a condition of a query. A condition can evaluate to TRUE or FALSE. It contains a match key that identifies what to match, a match value that identifies what to match with, and a match type that identifies how to match.

```

typedef struct _tag_FW_QUERY_CONDITION {
    FW_MATCH_KEY matchKey;
    FW_MATCH_TYPE matchType;
    FW_MATCH_VALUE matchValue;
} FW_QUERY_CONDITION,
*PFW_QUERY_CONDITION;

```

matchKey: This field identifies what information to match.

matchType: This field identifies how to perform the match operation.

matchValue: This field identifies what to match with.

A query condition structure MUST pass the following semantics checks:

- The **matchKey** field MUST have a valid **FW_MATCH_KEY** value that is less than FW_MATCH_KEY_MAX, MUST be a string of 1 or more characters, and MUST NOT be greater than or equal to 255 characters.
- The **matchType** field MUST have a valid **FW_MATCH_TYPE** value that is less than FW_MATCH_KEY_MAX.
- If the **matchType** field is equal to FW_MATH_TYPE_EQUAL, the **matchKey** field MUST be either FW_MATCH_KEY_GROUP or FW_MATCH_KEY_DIRECTION.
- If the **matchKey** field is equal to FW_MATCH_KEY_PROFILE or FW_MATCH_KEY_STATUS, the **matchValue** MUST have its type field equal to FW_DATA_TYPE_UINT32.
- If the **matchKey** field is equal to FW_MATCH_KEY_FILTERID, the **matchValue** MUST have its type field equal to FW_DATA_TYPE_UINT64.
- If the **matchKey** field is equal to FW_MATCH_KEY_PROTOCOL, FW_MATCH_KEY_LOCAL_PORT, or FW_MATCH_KEY_REMOTE_PORT; then the **matchValue** MUST have its type field equal to FW_DATA_TYPE_UINT16.
- If the **matchKey** field is equal to FW_MATCH_KEY_OBJECTID, FW_MATCH_KEY_APP_PATH, FW_MATCH_KEY_GROUP, or FW_MATCH_KEY_SVC_NAME; then the **matchValue** MUST have its type field equal to FW_DATA_TYPE_UNICODE_STRING.

2.2.84 FW_QUERY_CONDITIONS

This structure is used to contain a number of [FW_QUERY_CONDITION](#) elements. This structure can evaluate to either TRUE or FALSE. It evaluates to TRUE if all query condition elements evaluate to TRUE; otherwise, it evaluates to FALSE.

```

typedef struct _tag_FW_QUERY_CONDITIONS {
    unsigned LONG dwNumEntries;
    [size_is(dwNumEntries)] FW_QUERY_CONDITION* pAndedConditions;
} FW_QUERY_CONDITIONS,
*PFW_QUERY_CONDITIONS;

```

dwNumEntries: Specifies the number of query conditions that the structure contains.

pAndedConditions: A pointer to an array of FW_QUERY_CONDITIONS elements, which are all logically AND'd together. The number of elements is given by dwNumEntries.

A query condition structure MUST pass the following semantic checks:

- If the **dwNumEntries** field is zero, the **AndedConditions** field MUST be NULL; and if the **dwNumEntries** field is not zero, the **AndedConditions** field MUST NOT be NULL.
- If the **AndedConditions** field array has a **FW_QUERY_CONDITION** element with the **matchKey** field equal to FW_MATCH_KEY_LOCAL_PORT or FW_MATCH_KEY_REMOTE_PORT at position N of the array, the array MUST have another element whose **matchKey** field is equal to FW_MATCH_KEY_PROTOCOL at position M, where M < N.
- All elements of the **AndedConditions** array MUST have valid **FW_QUERY_CONDITION** structures.

2.2.85 FW_QUERY

This structure is used to query objects from the store. The structure contains a number of [FW_QUERY_CONDITIONS](#) elements. This structure can evaluate to either TRUE or FALSE. It evaluates to TRUE if at least one of the query conditions containers evaluates to TRUE; otherwise, if all evaluate to FALSE, it evaluates to FALSE.

```
typedef struct _tag_FW_QUERY {
    unsigned SHORT wSchemaVersion;
    unsigned LONG dwNumEntries;
    [size_is(dwNumEntries)] FW_QUERY_CONDITIONS* ORConditions;
    FW_RULE_STATUS Status;
} FW_QUERY,
*PFW_QUERY;
```

wSchemaVersion: The schema version of the query object. The version MUST be at least 0x00020A.

dwNumEntries: This field specifies the number of query conditions containers that the structure contains.

ORConditions: A pointer to an array of **FW_QUERY_CONDITIONS** elements, which are all logically OR'd together. The number of elements is given by **dwNumEntries**.

Status: The status code of the query, as specified by the [FW_RULE_STATUS](#) enumeration. This field is filled out when the structure is returned as output. On input, this field SHOULD be set to FW_RULE_STATUS_OK.

The following are semantic checks that query object MUST pass:

- The **wSchemaVersion** MUST NOT be less than 0x00020A.
- If the **dwNumEntries** field is zero, the **ORConditions** field MUST be NULL, and if the **dwNumEntries** field is not zero, the **ORConditions** field MUST NOT be NULL.
- The **ORConditions** field MUST have valid **FW_QUERY_CONDITIONS** elements.
- If the query object is used for querying connection security rules, it MUST NOT have any conditions with **matchKey** equal to FW_MATCH_KEY_APP_PATH or FW_MATCH_KEY_SVC_NAME.

- If the query object is being used for querying main mode rules, it MUST NOT have any conditions with **matchKey** equal to FW_MATCH_KEY_PROTOCOL, FW_MATCH_KEY_LOCAL_PORT, FW_MATCH_KEY_REMOTE_PORT, FW_MATCH_KEY_GROUP, or FW_MATCH_KEY_DIRECTION, or any of the match keys disallowed by connection security rules.
- If the query object is being used for querying authentication or cryptographic sets, it MUST NOT have any conditions with **matchKey** equal to FW_MATCH_KEY_PROFILE or FW_MATCH_KEY_FILTERID, or any of the match keys disallowed by main mode rules.

2.2.86 FW_POLICY_STORE_HANDLE

```
typedef [context_handle] void* FW_POLICY_STORE_HANDLE;  
  
typedef [ref] FW_POLICY_STORE_HANDLE* PFW_POLICY_STORE_HANDLE;
```

This type is an RPC context handle. It is a handle to a policy store exposed by this protocol. This handle is used to manage the policy contained in each store. Policy stores are identified by the [FW_STORE_TYPE](#) enumeration.

2.2.87 FW_PRODUCT_HANDLE

This type is declared as follows:

```
typedef [context_handle] void* FW_PRODUCT_HANDLE;
```

This type is an RPC context handle. It is a handle to the third-party software components that are registered with the firewall and advanced security component which are exposed through this protocol.

3 Protocol Details

The client side of this protocol is simply a pass-through. That is, there are no additional timers or other states required on the client side of this protocol. Calls made by the higher-layer protocol or application are passed directly to the transport, and the results returned by the transport are passed directly back to the higher-layer protocol or application.

3.1 Server Details

3.1.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with what is described in this document.

The server maintains several policy stores, one for each policy store type supported over the wire. More precisely, `FW_STORE_TYPE_NOT_USED_VALUE_3` and `FW_STORE_TYPE_NOT_USED_VALUE_4` are not maintained. All of these stores are read, merged, and enforced. The local and dynamic stores are online stores, which means that a write operation done in the store is immediately reflected in the enforcement of its traffic. Descriptions of the policy stores can be found in [FW STORE TYPE \(section 2.2.1\)](#). Also, please note that the order in which stores are loaded does not matter.

A store is composed of the following components: [global policy configuration options \(section 2.2.41\)](#), [profile configuration options \(section 2.2.37\)](#), and policy objects. The global configuration options are easily represented by a set of variables to contain each of the global configuration option values. These variables should have the types corresponding to the global configurations.

The store contains one profile configuration option component for each profile type supported. The profile configuration option component is also represented by a set of variables that contains each of the profile configuration option values. There is one variable for each option, and each of them has the corresponding option type.

There are five types of policy objects: firewall rules, connection security rules, authentication sets, crypto sets, and main mode rules, which are defined in this protocol as the data types [FW RULE \(section 2.2.36\)](#), [FW_CS_RULE \(section 2.2.53\)](#), [FW_AUTH_SET \(section 2.2.57\)](#), [FW_CRYPT_SET \(section 2.2.66\)](#), and [FW_MM_RULE \(section 2.2.77\)](#), respectively. The connection security rules include references to authentication sets and crypto sets in the store that can be enforced. To maintain the policy objects, a store contains four linked lists, one for each object type. These structures maintain the relationship between the policy object and the enforcement state objects and/or current enforcement state within the store. Enforcement state is the collection of data that includes enforcement state objects plus other state data required by the implementation. The type of data that represents the state data in the enforcement state and the enforcement state objects are implementation specific. Enforcement state objects are implementation-specific objects that can be added, deleted, or modified, and do the job of enforcing actions on the traffic in the network stack. Depending on the implementation, they might or might not need to be included in the enforcement state of each policy object. <15>

Each server maintains a copy of the store in memory as well as a copy of the store on disk. Write operations performed with the methods that follow affect both memory and disk representations. When dealing with online stores, the write operations also affect the enforcement and the state needed to perform this enforcement. When the write operation completes successfully on an online

store, the caller can assume that enforcement of the policy change is already happening. Implementations should also guarantee that policy changes, coming from any update methods, are synchronized and that calls see one consistent policy snapshot at the time of the call. Callers do not need to see the same policy snapshot across calls. Policy snapshot refers to the policies contained and the enforcement state associated with it. Implementations do not need to provide transactional support on each call; so even though calls are synchronized, failures between calls can occur without a rollback.

3.1.2 Timers

No protocol timer events are required on the server side other than the timers required by the underlying RPC transport, as specified in [\[MS-RPCE\]](#).

3.1.3 Initialization

The server initializes when the server host machine starts. At that time, the server reads the disk representation of the stores into the memory representation of the policy stores. Please see section [2.2.1](#) for the definition of stores. A store is composed of the following components: global policy configuration options (section [2.2.41](#)), profile configuration options (section [2.2.37](#)), and policy objects. Please note that the order of all maintained stores defined in section [2.2.1](#) does not matter. The server reads configuration option values into its memory variables and does the same with the profile configuration options. It then reads all the policy objects, validating each of them and storing them in linked lists.

From the perspective of this protocol, only locally stored data is retrieved and reported. The location and storage of data can be implementation-specific. How the data gets to the chosen location is also implementation-specific and orthogonal to this protocol. When the firewall and advanced security component initializes, whatever Group Policy data has already been downloaded to the server computer host is read, enforced, and reported through this protocol. The server computer host gets notified of Group Policy updates, and notifies the firewall and advanced security component to read the policy store and enforce the policy and be ready to report the new policy through this protocol. Any race conditions are handled by the logic described in the last two sentences, together with the synchronization guarantees described in section [3.1.1](#). Group Policies are downloaded to the host computer using [Group Policy: Registry Extension Encoding](#) (for more information, see [\[MS-GPREG\]](#)). For more information on this Group Policy download mechanism (such as timings, reliability, or how long it takes for a policy change to propagate across a network), see [\[MS-GPREG\]](#) sections [3.1.5.3](#) and [3.2.5.1](#). [\[MS-GPFAS\]](#) represents the contents of `FW_STORE_TYPE_GPO`. Clients apply the policy and learn what domain administrators are configured by using `FW_STORE_TYPE_GPO` (see section [2.2.1](#)).

The server then merges the stores. Each configuration option has a merge logic law, which determines how the merge should happen across stores. Examples of these are "Group policy wins" and "More secure wins". Policy objects are merged by simply enforcing all of the policy objects without considering collisions in rule IDs and set IDs. There is only one exception: default sets that are also merged because they always use the same set ID. The higher precedence default set wins and affects the connection security rules from all stores, which make a reference to a default set. Group Policy store has higher precedence.

The server then decides what profile is in use. It does so by analyzing the specific network adapters and the networks they are connected to.

The server then enforces the merged policy. While doing so, the server keeps enforcement state in each of the linked nodes of rules objects and sets objects, if appropriate. [<16>](#) This way, operations can use state to incrementally modify the needed policy and state, so that when the call is finished, the new policy is operational.

3.1.4 Message Processing Events and Sequencing Rules

This protocol MUST indicate to the RPC runtime that it is to perform a strict **Network Data Representation (NDR)** data consistency check at target level 6.0, as specified in [\[MS-RPCE\]](#).

This protocol MUST indicate to the RPC runtime, via the `strict_context_handle` attribute, that it is to reject the use of context handles that are created by using a different method of RPC interface than this one, as specified in [\[MS-RPCE\]](#) section 3.

Methods in RPC Opnum Order

Method	Description
RRPC_FWOpenPolicyStore	This method requests the server to open a specified policy store. Opnum: 0
RRPC_FWClosePolicyStore	This method receives an opened store handle and closes it, freeing any resources that were allocated by the server-to-server operations on the opened store. Opnum: 1
RRPC_FWRestoreDefaults	This method erases the local policy store and replaces it with the default policy that the server host had out of the box after installation. After the method returns, the local store contains exactly the same policy as it did after installation. Opnum: 2
RRPC_FWGetGlobalConfig	This method retrieves the value of a global policy configuration option. The client specifies to the server from what store this value must be retrieved and in what specific configuration option it is interested. Opnum: 3
RRPC_FWSetGlobalConfig	This method modifies the value of a global policy configuration option. The client specifies to the server in what store this value must be written and what specific configuration option it is interested in modifying. Opnum: 4
RRPC_FWAddFirewallRule	This method requests the server to add the specified firewall rule in the policy contained in the policy store that is referenced by the specified opened policy store handle. Opnum: 5
RRPC_FWSetFirewallRule	This method requests the server to modify the specified firewall rule in the policy contained in the policy store that is referenced by the specified opened policy store handle. Opnum: 6

Method	Description
RRPC_FWDeleteFirewallRule	This method requests the server to delete the specified firewall rule in the policy contained in the policy store that is referenced by the specified opened policy store handle. Opnum: 7
RRPC_FWDeleteAllFirewallRules	This method deletes all firewall rules in the firewall linked list of the memory representation of the store being modified. Opnum: 8
RRPC_FWEnumFirewallRules	This method requests the server to return all the firewall rules contained in the store that is referenced by the <i>hPolicyStore</i> handle. The method returns a linked list of all the firewall rule objects. Opnum: 9
RRPC_FWGetConfig	This method retrieves the value of a profile configuration option. The client specifies to the server from what store and profile this value must be retrieved and in what specific configuration option it is interested. Opnum: 10
RRPC_FWSetConfig	This method modifies the value of a profile configuration option. The client specifies to the server in what store and profile this value must be written and what specific configuration option it is interested in modifying. Opnum: 11
RRPC_FWAddConnectionSecurityRule	This method requests the server to add the connection security rule in the policy contained in the policy store that is referenced by the specified opened policy store handle. Opnum: 12
RRPC_FWSetConnectionSecurityRule	This method requests the server to modify the specified connection security rule in the policy contained in the policy store that is referenced by the specified opened policy store handle. Opnum: 13
RRPC_FWDeleteConnectionSecurityRule	This method requests the server to delete the specified connection security rule in the policy contained in the policy store that is referenced by the specified opened policy store handle. Opnum: 14
RRPC_FWDeleteAllConnectionSecurityRules	This method requests the server to delete all the connection security rules in the policy contained in the policy store that is referenced by the specified opened policy store handle. Opnum: 15

Method	Description
RRPC_FWEnumConnectionSecurityRules	This method requests the server to return all the connection security rules contained in the store that is referenced by the <i>hPolicyStore</i> handle. The method returns a linked list of all the connection security rule objects. Opnum: 16
RRPC_FWAddAuthenticationSet	This method requests the server to add the authentication set in the policy contained in the policy store that is referenced by the specified opened policy store handle. Opnum: 17
RRPC_FWSetAuthenticationSet	This method requests the server to modify the specified authentication set in the policy contained in the policy store that is referenced by the specified opened policy store handle. Opnum: 18
RRPC_FWDeleteAuthenticationSet	This method requests the server to delete the specified authentication set in the policy contained in the policy store that is referenced by the specified opened policy store handle. Opnum: 19
RRPC_FWDeleteAllAuthenticationSets	This method requests the server to delete all the authentication sets of a specific IPsec phase in the policy contained in the policy store that is referenced by the specified opened policy store handle. Opnum: 20
RRPC_FWEnumAuthenticationSets	This method requests the server to return all the authentication sets of the specified IPsec phase contained in the store that is referenced by the <i>hPolicyStore</i> handle. The method returns a linked list of these objects. Opnum: 21
RRPC_FWAddCryptoSet	This method adds a cryptographic set in the cryptographic linked list of the memory representation of the store being modified. Opnum: 22
RRPC_FWSetCryptoSet	This method requests the server to modify the specified cryptographic set in the policy contained in the policy store that is referenced by the specified opened policy store handle. Opnum: 23
RRPC_FWDeleteCryptoSet	This method requests the server to delete the specified cryptographic set in the policy contained in the policy store that is referenced by the specified opened policy store handle. Opnum: 24

Method	Description
RRPC_FWDeleteAllCryptoSets	This method requests the server to delete all the cryptographic sets of a specific IPsec phase in the policy contained in the policy store that is referenced by the specified opened policy store handle. Opnum: 25
RRPC_FWEnumCryptoSets	This method requests the server to return all the cryptographic sets of the specified IPsec phase contained in the store that is referenced by the <i>hPolicyStore</i> handle. The method returns a linked list of all these cryptographic objects. Opnum: 26
RRPC_FWEnumPhase1SAs	This method requests the server to return all the security associations of the IPsec first-negotiation phase contained in the store that is referenced by the <i>hPolicyStore</i> handle. The method returns a linked list of all these security associations. Opnum: 27
RRPC_FWEnumPhase2SAs	This method requests the server to return all the security associations of the IPsec second-negotiation phase contained in the store that is referenced by the <i>hPolicyStore</i> handle. The method returns a linked list of all these security associations. Opnum: 28
RRPC_FWDeletePhase1SAs	This method requests the server to delete all the IPsec first negotiation phase security associations that match the specified endpoints. Opnum: 29
RRPC_FWDeletePhase2SAs	This method requests the server to delete all the IPsec second negotiation phase security associations that match the specified endpoints. Opnum: 30
RRPC_FWEnumProducts	This method requests the server to return all the registered third-party software components registered with the firewall and advanced security component. Opnum: 31
RRPC_FWAddMainModeRule	This method requests the server to add the main mode rule in the policy contained in the policy store that is referenced by the specified opened policy store handle. Opnum: 32
RRPC_FWSetMainModeRule	This method requests the server to modify the specified main mode rule in the policy contained in the policy store that is referenced by the specified opened policy store handle. Opnum: 33

Method	Description
RRPC_FWDeleteMainModeRule	This method requests the server to delete the specified main mode rule in the policy contained in the policy store that is referenced by the specified opened policy store handle. Opnum: 34
RRPC_FWDeleteAllMainModeRules	This method requests the server to delete all the main mode rules in the policy contained in the policy store that is referenced by the specified opened policy store handle. Opnum: 35
RRPC_FWEnumMainModeRules	This method requests the server to return all the main mode rules contained in the store that is referenced by the <i>hPolicyStore</i> handle. The method returns a linked list of all the main mode rule objects. Opnum: 36
RRPC_FWQueryFirewallRules	This method requests the server to return all the firewall rules that match the specified query object contained in the store that is referenced by the <i>hPolicyStore</i> handle. The method returns a linked list of all the firewall rule objects. Opnum: 37
RRPC_FWQueryConnectionSecurityRules	This method requests the server to return all the connection security rules that match the specified query object contained in the store that is referenced by the <i>hPolicyStore</i> handle. The method returns a linked list of all the connection security rule objects. Opnum: 38
RRPC_FWQueryMainModeRules	This method requests the server to return all the main mode rules that match the specified query object contained in the store that is referenced by the <i>hPolicyStore</i> handle. The method returns a linked list of all the main mode rule objects. Opnum: 39
RRPC_FWQueryAuthenticationSets	This method requests the server to return all the authentication sets that match the specified query object contained in the store that is referenced by the <i>hPolicyStore</i> handle. The method returns a linked list of all the authentication set objects. Opnum: 40
RRPC_FWQueryCryptoSets	This method requests the server to return all the crypto sets that match the specified query object contained in the store that is referenced by the <i>hPolicyStore</i> handle. The method returns a linked list of all the crypto set objects. Opnum: 41
RRPC_FWEnumNetworks	This method requests the server to return all the networks to which the host that has the firewall and

Method	Description
	advanced security component is connected. Opnum: 42
RRPC FWEnumAdapters	This method requests the server to return all the network interfaces that are used by the host that has the firewall and advanced security component. Opnum: 43
RRPC FWGetGlobalConfig2 10	This method retrieves the value of a global policy configuration option. The client specifies to the server from what store this value must be retrieved and in what specific configuration option it is interested. Opnum: 44
RRPC FWGetConfig2 10	This method retrieves the value of a profile configuration option. The client specifies to the server from what store and profile this value must be retrieved and in what specific configuration option it is interested. Opnum: 45
RRPC FWAddFirewallRule2 10	This method requests the server to add the specified firewall rule in the policy contained in the policy store that is referenced by the specified opened policy store handle. Opnum: 46
RRPC FWSetFirewallRule2 10	This method requests the server to modify the specified firewall rule in the policy contained in the policy store that is referenced by the specified opened policy store handle. Opnum: 47
RRPC FWEnumFirewallRules2 10	This method requests the server to return all the firewall rules contained in the store that is referenced by the hPolicyStore handle. The method returns a linked list of all the firewall rule objects. Opnum: 48
RRPC FWAddConnectionSecurityRule2 10	This method requests the server to add the connection security rule in the policy contained in the policy store that is referenced by the specified opened policy store handle. Opnum: 49
RRPC FWSetConnectionSecurityRule2 10	This method requests the server to modify the specified connection security rule in the policy contained in the policy store that is referenced by the specified opened policy store handle. Opnum: 50
RRPC FWEnumConnectionSecurityRules2 10	This method requests the server to return all the connection security rules contained in the store that is referenced by the hPolicyStore handle. The method

Method	Description
	returns a linked list of all the connection security rule objects. Opnum: 51
RRPC_FWAddAuthenticationSet2_10	This method requests the server to add the authentication set in the policy contained in the policy store that is referenced by the specified opened policy store handle. Opnum: 52
RRPC_FWSetAuthenticationSet2_10	This method requests the server to modify the specified authentication set in the policy contained in the policy store that is referenced by the specified opened policy store handle. Opnum: 53
RRPC_FWEnumAuthenticationSets2_10	This method requests the server to return all the authentication sets of the specified IPsec phase contained in the store that is referenced by the <i>hPolicyStore</i> handle. The method returns a linked list of these objects. Opnum: 54
RRPC_FWAddCryptoSet2_10	This method adds a cryptographic set in the cryptographic linked list of the memory representation of the store being modified. Opnum: 55
RRPC_FWSetCryptoSet2_10	This method requests the server to modify the specified cryptographic set in the policy contained in the policy store that is referenced by the specified opened policy store handle. Opnum: 56
RRPC_FWEnumCryptoSets2_10	This method requests the server to return all the cryptographic sets of the specified IPsec phase that is contained in the store that is referenced by the <i>hPolicyStore</i> handle. The method returns a linked list of all these cryptographic objects. Opnum: 57

3.1.4.1 RRPC_FWOpenPolicyStore (Opnum 0)

The **RRPC_FWOpenPolicyStore** method requests the server to open a specified policy store. The store can be opened for reading or for editing the firewall policy. The method also provides a handle to the opened store with which the client can then perform operations on this policy store. The server keeps a reference to the binary version passed by the client associated with the returned handle.

```

unsigned long RRPC_FWOpenPolicyStore(
    [in] FW_CONN_HANDLE rpcConnHandle,
    [in] unsigned short BinaryVersion,
    [in, range(FW_STORE_TYPE_INVALID+1, FW_STORE_TYPE_MAX-1)]
        FW_STORE_TYPE StoreType,

```

```

[in, range(FW_POLICY_ACCESS_RIGHT_INVALID+1, FW_POLICY_ACCESS_RIGHT_MAX-1)]
    FW_POLICY_ACCESS_RIGHT AccessRight,
[in] unsigned long dwFlags,
[out] PFW_POLICY_STORE_HANDLE phPolicyStore
);

```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

BinaryVersion: This parameter specifies the RPC interface binary version. This implies versions of the methods and versions of the structures. This value **MUST** be a valid Protocol Version (see section [1.7](#)). See section [1.7](#) for capability negotiation based on the BinaryVersion.

StoreType: This parameter specifies the policy store type that the client wants to open.

AccessRight: This parameter specifies the read or read/write access rights that the client is requesting on the store.

dwFlags: This parameter specifies flags that can modify the behavior that other methods might have when called with the returned store handle. This parameter is reserved for future use; callers **SHOULD** pass a value of zero on the wire.

phPolicyStore: This is an output parameter that provides a pointer to an [FW_POLICY_STORE_HANDLE](#) data type. If successful, this parameter contains a handle to the opened store.

Return Values: The method returns 0 if successful; if failed, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#).

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server **MUST** validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.2 RRPC_FWClosePolicyStore (Opnum 1)

The **RRPC_FWClosePolicyStore** method receives an opened store handle, closes it, and deallocates the handle provided to the client.

```

unsigned long RRPC_FWClosePolicyStore(
    [in] FW_CONN_HANDLE rpcConnHandle,
    [in, out] PFW_POLICY_STORE_HANDLE phPolicyStore
);

```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

phPolicyStore: This is an input and output parameter that provides a pointer to an [FW_POLICY_STORE_HANDLE](#) data type. The data type **MUST** contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method, which the client intends to stop using and close.

Return Values: The method returns 0 if successful; if failed, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#).

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.3 RRPC_FWRestoreDefaults (Opnum 2)

The **RRPC_FWRestoreDefaults** method erases the local policy store and replaces it with the default policy that the server host had out of the box after installation. After the method returns, the local store contains exactly the same policy it did after installation.

```
unsigned long RRPC_FWRestoreDefaults(  
    [in] FW_CONN_HANDLE rpcConnHandle  
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

Return Values: The method returns 0 if successful; if failed, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#).

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

This method simply erases the memory and disk representation of the local store, and then copies the default settings into the disk representation of the default store, and then re-executes part of the initialization process defined in section [3.1.3](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.4 RRPC_FWGetGlobalConfig (Opnum 3)

The **RRPC_FWGetGlobalConfig** method retrieves the value of a global policy configuration option. The client specifies to the server from what store this value MUST be retrieved and in what specific configuration option it is interested.

```
unsigned long RRPC_FWGetGlobalConfig(  
    [in] FW_CONN_HANDLE rpcConnHandle,  
    [in] unsigned short BinaryVersion,  
    [in] FW_STORE_TYPE StoreType,  
    [in, range(FW_GLOBAL_CONFIG_INVALID+1, FW_GLOBAL_CONFIG_MAX-1)]  
        FW_GLOBAL_CONFIG configID,  
    [in] unsigned long dwFlags,  
    [in, out, unique, size_is(cbData), length_is(*pcbTransmittedLen)]  
        unsigned char* pBuffer,  
    [in] unsigned long cbData,  
    [in, out] unsigned long* pcbTransmittedLen,  
    [out] unsigned long* pcbRequired  
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

BinaryVersion: This parameter specifies the RPC interface binary version. This implies versions of the methods and versions of the structures.

StoreType: This parameter specifies the policy store from which the client wants to retrieve the configuration option value.

configID: This parameter specifies the specific global policy configuration option the client is interested in retrieving.

dwFlags: This parameter is a combination of flags from the [FW_CONFIG_FLAGS](#) enumeration, which modifies the behavior of this method, as specified in the definition of the enumeration.

pBuffer: This is an input/output parameter. This parameter is a pointer to the buffer that the client provides to contain the value of the profile configuration option being requested.

cbData: This parameter is the size of the buffer that the *pBuffer* parameter points to.

pcbTransmittedLen: This is a pointer to an input and output parameter that specifies the length of the transmitted data within the buffer.

pcbRequired: This is a pointer to an output parameter that specifies the required minimum buffer size in octets in order for the method to be able to return the configuration value. This output parameter is nonzero only if the buffer (pointed to by *pBuffer* and whose size is *cbData*) was not big enough to contain the value.

Return Values: The method returns 0 if successful; if failed, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The client does not have the required credentials to call the method.
0x00000002 ERROR_FILE_NOT_FOUND	The specific configuration option is not found within the policy. This means that it is not configured. If the option is not configured in any other store, the firewall uses a default value.
0x00000032 ERROR_NOT_SUPPORTED	The store type specified does not support this method.
0x000000EA ERROR_MORE_DATA	The buffer is not big enough to hold the configuration option value.
0x00000057 ERROR_INVALID_PARAMETER	One of the parameters of this method is incorrect, or is required and not specified. This error can be returned because: <ul style="list-style-type: none">▪ The specific configuration option is not meant to be available in the specified store.▪ The specified configuration option is not defined.▪ One of the required values is not specified.▪ The buffer size is not enough to hold the specific value.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.5 RRPC_FWSetGlobalConfig (Opnum 4)

The **RRPC_FWSetGlobalConfig** method modifies the value of a global policy configuration option. The client specifies to the server in what store this value MUST be written and what specific configuration option it is interested in modifying.

```
unsigned long RRPC_FWSetGlobalConfig(  
    [in] FW_CONN_HANDLE rpcConnHandle,  
    [in] unsigned short BinaryVersion,  
    [in] FW_STORE_TYPE StoreType,  
    [in, range(FW_GLOBAL_CONFIG_INVALID+1, FW_GLOBAL_CONFIG_MAX-1)]  
        FW_GLOBAL_CONFIG configID,  
    [in, unique, size_is(dwBufSize)]  
        unsigned char* lpBuffer,  
    [in, range(0, 10*1024)] unsigned long dwBufSize  
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

BinaryVersion: This parameter specifies the RPC interface binary version. This implies versions of the methods and versions of the structures.

StoreType: This parameter specifies the policy store in which the client wants to modify this configuration option.

configID: This parameter specifies the specific global policy configuration option the client wants to modify.

lpBuffer: This is an input parameter. This parameter is a pointer to the buffer that the client provides containing the value to write on the configuration option specified. If the buffer is NULL, this method deletes the configuration option.

dwBufSize: This parameter is the size of the buffer to which the *pBuffer* parameter points.

Return Values: The method returns 0 if successful; if failed, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The client does not have the required credentials to call the method.
0x00000032 ERROR_NOT_SUPPORTED	The store type specified does not support this method.
0x00000057 ERROR_INVALID_PARAMETER	One of the parameters of this method is incorrect, or is required and not specified. This error can be returned because: <ul style="list-style-type: none">▪ The specific configuration option is not meant to be available in the specified store.

Return value/code	Description
	<ul style="list-style-type: none"> ▪ The specified configuration option is not defined. ▪ One of the required values is not specified. ▪ The buffer is null but <i>dwBufSize</i> says otherwise. ▪ The buffer size is not enough to hold the specific value.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

This method performs a merge operation of the resultant configuration values, as defined in section [3.1.3](#). It then determines what modifications are necessary on the rule objects to make sure the policy is enforced.

The server **MUST** validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.6 RRPC_FWAddFirewallRule (Opnum 5)

The **RRPC_FWAddFirewallRule** method requests the server to add the specified firewall rule in the policy contained in the policy store that is referenced by the handle specified in the *hPolicyStore* parameter.

```
ULONG RRPC_FWAddFirewallRule(
    [in] FW_CONN_HANDLE rpcConnHandle,
    [in] FW_POLICY_STORE_HANDLE hPolicyStore,
    [in] PFW_RULE2_0 pRule
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicyStore: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type **MUST** contain an opened policy store handle that is successfully opened by using the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle **MUST** have read/write access rights.

pRule: This parameter represents the firewall rule that the client wants to add to the store. The rule **MUST** be a valid rule, as specified in the definition of the [FW_RULE2_0](#) data type.

Return Values: The method returns 0 if successful; if it fails, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x000000B7 ERROR_ALREADY_EXISTS	The specified rule has a rule ID that already exists in the specified store.
0x00000032 ERROR_NOT_SUPPORTED	The specified store does not support this method; the store might be read-only.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The <i>hPolicyStore</i> handle was not opened with read/write access rights. The error is also returned if the client does not have the required credentials to call the method.
0x00000057 ERROR_INVALID_PARAMETER	A parameter of this method is incorrect, or is required and not specified. This error can be returned because: <ul style="list-style-type: none"> ▪ The <i>pRule</i> object did not pass the firewall rule validations that are specified in the definition of the FW_RULE data type. ▪ One of the required values is not specified. ▪ A policy store does not support rules with profile conditions other than ALL profiles. ▪ The wszLocalApplication field of the rule contains a string that was determined to be an invalid path. <17>

Exceptions Thrown: No exceptions are thrown except those that are thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

This method adds a firewall rule to the firewall linked list of the memory representation of the store being modified. It also writes through and saves the rule in disk. If called on an online store, the firewall rule is also enforced.

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.7 RRPC_FWSetFirewallRule (Opnum 6)

The **RRPC_FWSetFirewallRule** method requests the server to modify the specified firewall rule in the policy contained in the policy store that is referenced by the handle specified in the *hPolicyStore* parameter.

```
ULONG RRPC_FWSetFirewallRule(
    [in] FW_CONN_HANDLE rpcConnHandle,
    [in] FW_POLICY_STORE_HANDLE hPolicyStore,
    [in] PFW_RULE2_0 pRule
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicyStore: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle that is successfully opened by using the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read/write access rights.

pRule: This parameter represents the firewall rule that the client wants to modify in the store. The rule MUST be a valid rule, as specified in the definition of the [FW_RULE2_0](#) data type.

Return Values: The method returns 0 if successful; if it fails, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following are common.

Return value/code	Description
0x00000032 ERROR_NOT_SUPPORTED	The specified store does not support this method; the store might be read-only.
0x00000005 ERROR_ACCESS_DENIED	The <i>hPolicyStore</i> handle was not opened with read/write access rights. The error is also returned if the client does not have the required credentials to call the method.
0x00000002 ERROR_FILE_NOT_FOUND	The specified rule that is referenced by the wszRuleID member string of the FW_RULE data type is not found in the policy store.
0x00000057 ERROR_INVALID_PARAMETER	A parameter of this method is incorrect, or is required and not specified. This error can be returned because: <ul style="list-style-type: none"> ▪ The <i>pRule</i> object did not pass the firewall rule validations that are specified in the definition of the FW_RULE data type. ▪ One of the required values is not specified. ▪ A policy store does not support rules that have profile conditions other than ALL profiles. ▪ The wszLocalApplication field of the rule contains a string that was determined to be an invalid path.

Exceptions Thrown: No exceptions are thrown except those that are thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.8 RRPC_FWDeleteFirewallRule (Opnum 7)

The **RRPC_FWDeleteFirewallRule** method requests the server to delete the specified firewall rule in the policy contained in the policy store referenced by the handle specified in the *hPolicyStore* parameter.

```
unsigned long RRPC_FWDeleteFirewallRule(
    [in] FW_CONN_HANDLE rpcConnHandle,
    [in] FW_POLICY_STORE_HANDLE hPolicyStore,
    [in, string, ref] const wchar_t* wszRuleID
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicyStore: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read/write access rights.

wszRuleID: This parameter is the pointer to a string that is the ID of the firewall rule the client wants to delete from the specified store.

Return Values: The method returns 0 if successful; if failed, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following are common.

Return value/code	Description
0x00000032 ERROR_NOT_SUPPORTED	The specified store does not support this method; the store might be read-only.
0x00000005 ERROR_ACCESS_DENIED	The <i>hPolicyStore</i> handle was not opened with read/write access rights. The error is also returned if the client does not have the required credentials to call the method.
0x00000002 ERROR_FILE_NOT_FOUND	The specified rule referenced by the wszRuleID member string of the FW_RULE data type is not found in the policy store.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

This method deletes a firewall rule already stored in the firewall linked list of the memory representation of the store being modified. It uses this list to determine if the rule exists or not. It also writes through and deletes the rule from disk. If called on an online store, the removal of the firewall rule is also enforced.

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.9 RRPC_FWDeleteAllFirewallRules (Opnum 8)

The **RRPC_FWDeleteAllFirewallRules** method deletes all firewall rules in the firewall linked list of the memory representation of the store being modified. It also writes through and deletes all rules from the disk representation. If called on an online store, no firewall rules are enforced after the method returns.

```
unsigned long RRPC_FWDeleteAllFirewallRules(
    [in] FW_CONN_HANDLE rpcConnHandle,
    [in] FW_POLICY_STORE_HANDLE hPolicyStore
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicyStore: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read/write access rights.

Return Values: The method returns 0 if successful; if failed, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following are common.

Return value/code	Description
0x00000032 ERROR_NOT_SUPPORTED	The specified store does not support this method; the store might be read-only.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The <i>hPolicyStore</i> handle was not opened with read/write access rights. The error is also returned if the client does not have the required credentials to call the method.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.10 RRPC_FWEnumFirewallRules (Opnum 9)

The **RRPC_FWEnumFirewallRules** method requests the server to return all the firewall rules contained in the store that is referenced by the *hPolicyStore* handle. The method returns a linked list of all the firewall rule objects.

```

ULONG RRPC_FWEnumFirewallRules(
    [in] FW_CONN_HANDLE rpcConnHandle,
    [in] FW_POLICY_STORE_HANDLE hPolicyStore,
    [in] ULONG dwFilteredByStatus,
    [in] ULONG dwProfileFilter,
    [in] USHORT wFlags,
    [out, ref] ULONG* pdwNumRules,
    [out] PFW_RULE2_0* ppRules
);

```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicyStore: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle that is successfully opened by using the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read or read/write access rights.

dwFilteredByStatus: This parameter is a combination of flags from the [FW_RULE_STATUS_CLASS](#) enumeration. This method uses this bitmask to determine whether rules should be returned. Rules that contain a status code from the class specified by this parameter will be returned in the linked list.

dwProfileFilter: This parameter is a combination of flags from the [FW_PROFILE_TYPE](#) enumeration. This method also uses this parameter to determine whether rules should be returned. Rules that contain a profile specified by this parameter will be returned in the linked list.

wFlags: This parameter is a combination of flags from the [FW_ENUM_RULES_FLAGS](#) enumeration, which modifies the behavior of the method and performs operations on the rules before returning them in the linked list.

pdwNumRules: This output parameter, if successful, MUST be equal to the number of rules returned.

ppRules: This output parameter, if successful, contains a linked list of [FW_RULE2_0](#) data types.

Return Values: The method returns 0 if successful; if it fails, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following are common.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The <i>hPolicyStore</i> handle was not opened with read/write access rights. The error is also returned if the client does not have the required credentials to call the method.
0x00000057 ERROR_INVALID_PARAMETER	The <i>dwProfileFilter</i> parameter contains profiles that are not valid.

Exceptions Thrown: No exceptions are thrown except those that are thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.11 RRPC_FWGetConfig (Opnum 10)

The **RRPC_FWGetConfig** method retrieves the value of a profile configuration option. The client specifies to the server from what store and profile this value MUST be retrieved and in what specific configuration option it is interested.

```

unsigned long RRPC_FWGetConfig(
    [in] FW_CONN_HANDLE rpcConnHandle,
    [in] FW_POLICY_STORE_HANDLE hPolicyStore,
    [in, range(FW_PROFILE_CONFIG_ENABLE_FW, FW_PROFILE_CONFIG_MAX-1)]
    FW_PROFILE_CONFIG configID,
    [in] FW_PROFILE_TYPE Profile,
    [in] unsigned long dwFlags,
    [in, out, unique, size_is(cbData), length_is(*pcbTransmittedLen)]
    unsigned char* pBuffer,
    [in] unsigned long cbData,
    [in, out] unsigned long* pcbTransmittedLen,
    [out] unsigned long* pcbRequired
);

```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicyStore: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read/write access rights.

configID: This parameter specifies the specific profile configuration option the client is interested in retrieving.

Profile: This parameter specifies from which specific profile this value MUST be retrieved.

dwFlags: This parameter is a combination of flags from the [FW_CONFIG_FLAGS](#) enumeration, which modifies the behavior of this method, as specified in the definition of the enumeration.

pBuffer: This is an input/output parameter. This parameter is a pointer to the buffer that the client provides to contain the value of the profile configuration option being requested.

cbData: This parameter is the size of the buffer that the *pBuffer* parameter points to.

pcbTransmittedLen: This is a pointer to an input and output parameter that specifies the length of the transmitted data within the buffer.

pcbRequired: This is a pointer to an output parameter that specifies the required minimum buffer size in octets for the method to be able to return the configuration value. This output parameter is nonzero only if the buffer (pointed to by *pBuffer* and whose size is *cbData*) was not big enough to contain the value.

Return Values: The method returns 0 if successful; if failed, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following are common.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The client does not have the required credentials to call the method.
0x00000002 ERROR_FILE_NOT_FOUND	The specific configuration option is not found within the policy. This means that it is not configured. If the option is not configured in any other store, the firewall uses a default value.
0x00000032 ERROR_NOT_SUPPORTED	The method does not support the specified combination of parameters. This can be because: <ul style="list-style-type: none">▪ The store type specified does not support this method.▪ The configuration option is not supported in this store.▪ The <i>Profile</i> parameter contains a combination of profiles (instead of a single profile) or an unknown profile.
0x000000EA ERROR_MORE_DATA	The buffer is not big enough to hold the configuration option value.
0x00000057 ERROR_INVALID_PARAMETER	One of the parameters of this method is incorrect, or is required and not specified. This error can be returned because: <ul style="list-style-type: none">▪ The specified configuration option is not defined.▪ One of the required values is not specified.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.12 RRPC_FWSetConfig (Opnum 11)

The **RRPC_FWSetConfig** method modifies the value of a profile configuration option. The client specifies to the server in what store and profile this value MUST be written and what specific configuration option it is interested in modifying.

```

unsigned long RRPC_FWSetConfig(
    [in] FW_CONN_HANDLE rpcConnHandle,
    [in] FW_POLICY_STORE_HANDLE hPolicyStore,
    [in, range(FW_PROFILE_CONFIG_ENABLE_FW, FW_PROFILE_CONFIG_MAX-1)]
        FW_PROFILE_CONFIG configID,
    [in] FW_PROFILE_TYPE Profile,
    [in, switch_is(configID)] FW_PROFILE_CONFIG_VALUE pConfig,
    [in, range(0, 10*1024)] unsigned long dwBufSize
);

```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicyStore: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read/write access rights.

configID: This parameter specifies the specific profile configuration option the client is interested in retrieving.

Profile: This parameter specifies from which specific profile this value MUST be retrieved.

pConfig: This is an input parameter. This parameter is a pointer to the buffer that the client provides containing the value to write on the configuration option specified. If the buffer is NULL, this method deletes the configuration option. The buffer is of type [FW_PROFILE_CONFIG_VALUE](#).

dwBufSize: This parameter is the size of the buffer that the *pConfig* parameter points to.

Return Values: The method returns 0 if successful; if failed, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000032 ERROR_NOT_SUPPORTED	The method does not support the specified combination of parameters. This can be because: <ul style="list-style-type: none"> The store type specified does not support this method. The Profile parameter contains a combination of profiles (instead of a single profile) or an unknown profile.
0x00000005 ERROR_ACCESS_DENIED	The <i>hPolicyStore</i> handle was not opened with read/write access rights. The error is also returned if the client does not have the required credentials to call the method.
0x00000057 ERROR_INVALID_PARAMETER	One of the parameters of this method is incorrect, or is required and not specified. This error can be returned because: <ul style="list-style-type: none"> The specific configuration option is not meant to be available in the specified store. The specified configuration option is not defined. The size of the buffer does not match the size of the type of

Return value/code	Description
	<p>the configuration value.</p> <ul style="list-style-type: none"> ▪ The buffer is null but <i>dwBufSize</i> says otherwise. ▪ The caller wants to set a LOG_MAX_FILE_SIZE that is not within the valid values [min, max]. ▪ The default action configuration value specifies a value that maps to neither allow nor block. ▪ The LOG_FILE_PATH configuration value contains the following invalid characters: /,*,?,",<,>, .

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

This method performs a merge operation of the resultant configuration values, as defined in section [3.1.3](#). It then determines what modifications are necessary on the rule objects (for example, remove rule enforcement if firewall is off) to make sure the policy is enforced.

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.13 RRPC_FWAddConnectionSecurityRule (Opnum 12)

The **RRPC_FWAddConnectionSecurityRule** method requests the server to add the connection security rule in the policy contained in the policy store that is referenced by the specified opened policy store handle.

```
ULONG RRPC_FWAddConnectionSecurityRule(
    [in] FW_CONN_HANDLE rpcConnHandle,
    [in] FW_POLICY_STORE_HANDLE hPolicy,
    [in] PFW_CS_RULE2_0 pRule
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle that is successfully opened by using the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read/write access rights.

pRule: This parameter represents the connection security rule that the client wants to add to the store. The rule MUST be a valid rule, as specified in the definition of the [FW_CS_RULE2_0](#) data type.

Return Values: This method returns 0 if successful; if it fails, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x000000B7 ERROR_ALREADY_EXISTS	The specified rule has a rule ID that already exists in the specified store.
0x00000032 ERROR_NOT_SUPPORTED	The specified store does not support this method; the store might be read-only.
0x00000005 ERROR_ACCESS_DENIED	The <i>hPolicy</i> handle was not opened with read/write access rights. The error is also returned if the client does not have the required credentials to call the method.
0x00000057 ERROR_INVALID_PARAMETER	A parameter of this method is incorrect, or is required and not specified. This error can be returned because: <ul style="list-style-type: none"> ▪ The <i>pRule</i> object did not pass the connection security rule validations specified in the definition of the FW_CS_RULE data type. ▪ The rule has a phase 2 crypto set that specified <code>FW_CRYPTOPRPTOCOL_AUTH_NO_ENCAP</code> (see section 2.2.61), and it is a tunnel mode rule, or it also has an AuthSet structure that specifies a preshared key auth method. ▪ A required value is not specified.

Exceptions Thrown: No exceptions are thrown except those that are thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

This method adds a connection security rule in the connection security link list of the memory representation of the store being modified. It also writes through and saves the rule to disk. If called on an online store, the connection security rule is also enforced.

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.14 RRPC_FWSetConnectionSecurityRule (Opnum 13)

The **RRPC_FWSetConnectionSecurityRule** method requests the server to modify the specified connection security rule in the policy contained in the policy store that is referenced by the handle specified in the *hPolicy* parameter.

```
ULONG RRPC_FWSetConnectionSecurityRule(
    [in] FW_CONN_HANDLE rpcConnHandle,
    [in] FW_POLICY_STORE_HANDLE hPolicy,
    [in] PFW_CS_RULE2_0 pRule
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle that is successfully opened by using the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read/write access rights.

pRule: This parameter represents the connection security rule that the client wants to modify in the store. The rule MUST be a valid rule, as specified in the definition of the [FW_CS_RULE2_0](#) data type.

Return Values: The method returns 0 if successful; if it fails, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000032 ERROR_NOT_SUPPORTED	The specified store does not support this method; the store might be read-only.
0x00000005 ERROR_ACCESS_DENIED	The <i>hPolicy</i> handle was not opened with read/write access rights. This error is also returned if the client does not have the required credentials to call the method.
0x00000002 ERROR_FILE_NOT_FOUND	The specified rule that is referenced by the <i>wszRuleID</i> member string of the FW_CS_RULE data type is not found in the policy store.
0x00000057 ERROR_INVALID_PARAMETER	A parameter of this method is incorrect, or is required and not specified. This error can be returned because: <ul style="list-style-type: none"> ▪ The <i>pRule</i> object did not pass the connection security rule validations that are specified in the definition of the FW_CS_RULE data type. ▪ The rule has a phase 2 crypto set that specified <code>FW_CRYPTOPRPTOCOL_AUTH_NO_ENCAP</code> (see section 2.2.61), and either it is a tunnel mode rule or it has an <code>AuthSet</code> that specifies a preshared key auth method. ▪ A required value is not specified.

Exceptions Thrown: No exceptions are thrown except those that are thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

This method modifies a connection security rule already stored in the connection security linked list of the memory representation of the store being modified. It uses this list to determine whether the rule exists. It also writes through and saves the rule in disk. If called on an online store, the connection security rule modifications are also enforced.

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.15 RRPC_FWDeleteConnectionSecurityRule (Opnum 14)

The **RRPC_FWDeleteConnectionSecurityRule** method requests the server to delete the specified connection security rule in the policy contained in the policy store referenced by the handle specified in the *hPolicy* parameter.

```

unsigned long RRPC_FWDeleteConnectionSecurityRule (
    [in] FW_CONN_HANDLE rpcConnHandle,
    [in] FW_POLICY_STORE_HANDLE hPolicy,
    [in, string, ref] wchar_t* pRuleId

```

);

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read/write access rights.

pRuleId: This parameter is the pointer to a string that is the ID of the connection security rule the client wants to delete from the specified store.

Return Values: The method returns 0 if successful; if failed, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000032 ERROR_NOT_SUPPORTED	The specified store does not support this method; the store might be read-only.
0x00000005 ERROR_ACCESS_DENIED	The <i>hPolicy</i> handle was not opened with read/write access rights. The error is also returned if the client does not have the required credentials to call the method.
0x00000002 ERROR_FILE_NOT_FOUND	The specified rule referenced by the <i>pRuleID</i> member string is not found in the policy store.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

This method deletes a connection security rule already stored in the connection security linked list of the memory representation of the store being modified. It uses this list to determine if the rule exists or not. It also writes through and deletes the rule from disk. If called on an online store, the removal of the connection security rule is also enforced.

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.16 RRPC_FWDeleteAllConnectionSecurityRules (Opnum 15)

The **RRPC_FWDeleteAllConnectionSecurityRules** method requests the server to delete all the connection security rules in the policy contained in the policy store referenced by the handle specified in the **hPolicy** parameter.

```
unsigned long RRPC_FWDeleteAllConnectionSecurityRules(  
    [in] FW_CONN_HANDLE rpcConnHandle,  
    [in] FW_POLICY_STORE_HANDLE hPolicy  
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read/write access rights.

Return Values: The method returns 0 if successful; if failed, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000032 ERROR_NOT_SUPPORTED	The specified store does not support this method; the store might be read-only.
0x00000005 ERROR_ACCESS_DENIED	The <i>hPolicy</i> handle was not opened with read/write access rights. The error is also returned if the client does not have the required credentials to call the method.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

This method deletes all connection security rules in the connection security linked list of the memory representation of the store being modified. It also writes through and deletes all rules from the disk representation. If called on an online store, no connection security rules are enforced after the method returns.

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.17 RRPC_FWEnumConnectionSecurityRules (Opnum 16)

The **RRPC_FWEnumConnectionSecurityRules** method requests the server to return all the connection security rules contained in the store that is referenced by the *hPolicy* handle. The method returns a linked list of all the connection security rule objects.

```
ULONG RRPC_FWEnumConnectionSecurityRules(
    [in] FW_CONN_HANDLE rpcConnHandle,
    [in] FW_POLICY_STORE_HANDLE hPolicy,
    [in] ULONG dwFilteredByStatus,
    [in] ULONG dwProfileFilter,
    [in] USHORT wFlags,
    [out, ref] ULONG* pdwNumRules,
    [out] PFW_CS_RULE2_0* ppRules
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle that is successfully opened by using the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read or read/write access rights.

dwFilteredByStatus: This parameter is a combination of flags from the [FW_RULE_STATUS_CLASS](#) enumeration. This method uses this bitmask to determine

whether rules should be returned. Rules that contain a status code from the class that is specified by this parameter will be returned in the linked list.

dwProfileFilter: This parameter is a combination of flags from the [FW_PROFILE_TYPE](#) enumeration. This method also uses this parameter to determine whether rules should be returned. Rules that contain a profile that is specified by this parameter will be returned in the linked list.

wFlags: This parameter is a combination of flags from the [FW_ENUM_RULES_FLAGS](#) enumeration, which modifies the behavior of the method and performs operations on the rules before returning them in the linked list.

pdwNumRules: This output parameter, if successful, MUST be equal to the number of rules returned.

ppRules: This output parameter, if successful, contains a linked list of [FW_CS_RULE2_0](#) data types.

Return Values: The method returns 0 if successful; if it fails, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The client does not have the required credentials to call the method.
0x00000057 ERROR_INVALID_PARAMETER	The <i>dwProfileFilter</i> parameter contains invalid profiles.

Exceptions Thrown: No exceptions are thrown except those that are thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.18 RRPC_FWAddAuthenticationSet (Opnum 17)

The **RRPC_FWAddAuthenticationSet** method requests the server to add the authentication set in the policy contained in the policy store referenced by the handle specified in the **hPolicy** parameter.

```
unsigned long RRPC_FWAddAuthenticationSet(  
    [in] FW_CONN_HANDLE rpcConnHandle,  
    [in] FW_POLICY_STORE_HANDLE hPolicy,  
    [in] PFW_AUTH_SET pAuth  
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read/write access rights.

pAuth: This parameter represents the authentication set the client wants to add to the store. The set MUST be valid, as specified in the definition of the [FW_AUTH_SET](#) data type.

Return Values: The method returns 0 if successful; if failed, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x000000B7 ERROR_ALREADY_EXISTS	The specified set has a set ID that already exists in the specified store.
0x00000032 ERROR_NOT_SUPPORTED	The specified store does not support this method; the store might be read-only.
0x00000005 ERROR_ACCESS_DENIED	The <i>hPolicy</i> handle was not opened with read/write access rights. The error is also returned if the client does not have the required credentials to call the method.
0x00000057 ERROR_INVALID_PARAMETER	One of the parameters of this method is incorrect, or is required and not specified. This error can be returned because: <ul style="list-style-type: none">▪ The <i>pAuth</i> object did not pass the authentication set validations specified in the definition of the FW_AUTH_SET data type.▪ One of the required values is not specified.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

This method adds an authentication set in the authentication linked list of the memory representation of the store being modified. It also writes through and saves the set in disk. If called on an online store and the set is a default set, the method enumerates the connection security rule list and reapplies each rule referencing this default set to complete the enforcement of the policy.

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.19 RRPC_FWSetAuthenticationSet (Opnum 18)

The **RRPC_FWSetAuthenticationSet** method requests the server to modify the specified authentication set in the policy contained in the policy store referenced by the handle specified in the *hPolicy* parameter.

```
unsigned long RRPC_FWSetAuthenticationSet(  
    [in] FW_CONN_HANDLE rpcConnHandle,  
    [in] FW_POLICY_STORE_HANDLE hPolicy,  
    [in] PFW_AUTH_SET pAuth  
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the

[RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read/write access rights.

pAuth: This parameter represents the authentication set the client wants to modify in the store. The set MUST be valid, as specified in the definition of the [FW_AUTH_SET](#) data type.

Return Values: The method returns 0 if successful; if failed, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000032 ERROR_NOT_SUPPORTED	The specified store does not support this method; the store might be read-only.
0x00000005 ERROR_ACCESS_DENIED	The <i>hPolicy</i> handle was not opened with read/write access rights. The error is also returned if the client does not have the required credentials to call the method.
0x00000002 ERROR_FILE_NOT_FOUND	The specified set referenced by the wszSetID member string of the FW_AUTH_SET data type is not found in the policy store.
0x00000057 ERROR_INVALID_PARAMETER	One of the parameters of this method is incorrect, or is required and not specified. This error can be returned because: <ul style="list-style-type: none">▪ The <i>pAuth</i> object did not pass the authentication set validations specified in the definition of the FW_AUTH_SET data type.▪ One of the required values is not specified.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

This method modifies an authentication set in the authentication linked list of the memory representation of the store being modified. It also writes through and saves the set in disk. If called on an online store, the method enumerates the connection security rules list and reapplies each rule referencing this default set to complete the enforcement of the policy.

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.20 RRPC_FWDeleteAuthenticationSet (Opnum 19)

The **RRPC_FWDeleteAuthenticationSet** method requests the server to delete the specified authentication set in the policy contained in the policy store referenced by the handle specified in the *hPolicy* parameter.

```
unsigned long RRPC_FWDeleteAuthenticationSet (  
    [in] FW_CONN_HANDLE rpcConnHandle,  
    [in] FW_POLICY_STORE_HANDLE hPolicy,  
    [in, range(FW_IPSEC_PHASE_INVALID+1, FW_IPSEC_PHASE_MAX-1)]  
        FW_IPSEC_PHASE IpSecPhase,  
    [in, string, ref] const wchar_t* wszSetId  
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read/write access rights.

IpSecPhase: This parameter specifies the IPsec negotiation phase type this set is used in.

wszSetId: This parameter is the pointer to a string that is the ID of the authentication set the client wants to delete from the specified store.

Return Values: The method returns 0 if successful; if failed, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000962 ERROR_ACTIVE_CONNECTIONS	The specified set is still referenced by connection security rules. This failure happens only when the set is not a default set. There is always a default set to use, either from other stores or a hard-coded one.
0x00000032 ERROR_NOT_SUPPORTED	The specified store does not support this method; the store might be read-only.
0x00000005 ERROR_ACCESS_DENIED	The <i>hPolicy</i> handle was not opened with read/write access rights. The error is also returned if the client does not have the required credentials to call the method.
0x00000002 ERROR_FILE_NOT_FOUND	The specified rule referenced by the wszSetID string is not found in the policy store.
0x00000057 ERROR_INVALID_PARAMETER	The specified IPsec phase is not a valid one.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

This method deletes an authentication set in the authentication linked list of the memory representation of the store being modified. It also writes through and saves the set in disk. If called on an online store, and the set is not a default set, the method does not delete the specified set if any connection rule references this set.

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.21 RRPC_FWDeleteAllAuthenticationSets (Opnum 20)

The **RRPC_FWDeleteAllAuthenticationSets** method requests the server to delete all the authentication sets of a specific IPsec phase in the policy contained in the policy store referenced by the handle specified in the *hPolicy* parameter.

```
unsigned long RRPC_FWDeleteAllAuthenticationSets(  
    [in] FW_CONN_HANDLE rpcConnHandle,  
    [in] FW_POLICY_STORE_HANDLE hPolicy,
```

```
[in, range(FW_IPSEC_PHASE_INVALID+1, FW_IPSEC_PHASE_MAX-1)]
    FW_IPSEC_PHASE IpSecPhase
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read/write access rights.

IpSecPhase: This parameter specifies the IPsec negotiation phase type in which this set is used.

Return Values: The method returns 0 if successful; if failed, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000962 ERROR_ACTIVE_CONNECTIONS	The specified set is still referenced by connection security rules. This failure happens only when the set is not a default set. There is always a default set to use, either from other stores or a hard-coded one.
0x00000032 ERROR_NOT_SUPPORTED	The specified store does not support this method; the store might be read-only.
0x00000005 ERROR_ACCESS_DENIED	The <i>hPolicy</i> handle was not opened with read/write access rights. The error is also returned if the client does not have the required credentials to call the method.
0x00000002 ERROR_FILE_NOT_FOUND	The specified rule referenced by the wszSetID string is not found in the policy store.
0x00000057 ERROR_INVALID_PARAMETER	The specified IPsec phase is not a valid one.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

This method deletes all the authentication sets in the authentication linked list of the memory representation of the store being modified. It also writes through and deletes the sets from disk. If called on an online store, the method does not delete the sets if any nondefault set is referenced by a connection security rule.

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.22 RRPC_FWEnumAuthenticationSets (Opnum 21)

The **RRPC_FWEnumAuthenticationSets** method requests the server to return all the authentication sets of the specified IPsec phase contained in the store referenced by the *hPolicy* handle. The method returns a linked list of these objects.

```

unsigned long RRPC_FWEnumAuthenticationSets(
    [in] FW_CONN_HANDLE rpcConnHandle,
    [in] FW_POLICY_STORE_HANDLE hPolicy,
    [in, range(FW_IPSEC_PHASE_INVALID+1, FW_IPSEC_PHASE_MAX-1)]
        FW_IPSEC_PHASE IpSecPhase,
    [in] unsigned long dwFilteredByStatus,
    [in] unsigned short wFlags,
    [out] unsigned long* pdwNumAuthSets,
    [out] PFW_AUTH_SET* ppAuth
);

```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read or read/write access rights.

IpSecPhase: This parameter specifies the specific IPsec negotiation phase to which this set applies.

dwFilteredByStatus: This parameter is a combination of flags from the [FW_RULE_STATUS_CLASS](#) enumeration. This method uses this bitmask to determine if rules should be returned or not. Sets that contain a status code of the class specified by this parameter will be returned in the linked list.

wFlags: This parameter is a combination of flags from the [FW_ENUM_RULES_FLAGS](#) that modifies the behavior of the method and performs operations on the sets before returning them in the linked list.

pdwNumAuthSets: This is an output parameter that on success MUST be equal to the number of sets returned.

ppAuth: This is an output parameter that on success contains a linked list of [FW_AUTH_SET](#) data types.

Return Values: The method returns 0 if successful; if failed, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The client does not have the required credentials to call the method.
0x00000057 ERROR_INVALID_PARAMETER	One of the parameters of this method is incorrect, or is required and not specified. This error can be returned because: <ul style="list-style-type: none"> ▪ The <i>IpSecPhase</i> parameter specifies an invalid IPsec negotiation phase. ▪ One of the required values is not specified.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

When this method is called, the server looks for the binary version of the client, which was associated with the *hPolicy* handle when the client sent the **RRPC_FWOpenPolicyStore()** call. The server compares this binary version parameter with the schema version it supports. If the server has a schema version of 0x0201 and the client passed a 0x0200 binary version, then the server removes all values that are not valid for a **FW_AUTH_SET** (section 2.2.57) structure that has a 0x0200 schema version. If the removed value was present on one or more suites of the set, the server removes those suites as a whole, leaving the remaining suites intact. For each set that had a value removed, the server sets a **FW_RULE_STATUS_PARTIALLY_IGNORED** value on the **Status** field of the set. Then the client receives authentication sets with values that correspond to the correct schema version, but the client recognizes that the information it has about the sets is potentially incomplete.

3.1.4.23 RRPC_FWAddCryptoSet (Opnum 22)

The **RRPC_FWAddCryptoSet** method adds a cryptographic set in the cryptographic linked list of the memory representation of the store being modified. It also writes through and saves the set to the disk. If called on an online store, and the set is a default set, the method enumerates the connection security rule list and reapplies each rule referencing this default set to complete the enforcement of the policy.

When the server is operating under normal security mode, the server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method. When the server is operating under common criteria mode, the server MUST NOT allow any callers and return an access denied error.

```
unsigned long RRPC_FWAddCryptoSet(  
    [in] FW_CONN_HANDLE rpcConnHandle,  
    [in] FW_POLICY_STORE_HANDLE hPolicy,  
    [in] PFW_CRYPTO_SET pCrypto  
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read/write access rights.

pCrypto: This parameter represents the cryptographic set the client wants to add to the store. The set MUST be valid, as specified in the definition of the [FW_CRYPTO_SET](#) data type.

Return Values: The method returns 0 if successful; if failed, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x000000B7 ERROR_ALREADY_EXISTS	The specified rule has a rule ID that already exists in the specified store.
0x00000032	The specified store does not support this method; the store might

Return value/code	Description
ERROR_NOT_SUPPORTED	be read-only.
0x00000005 ERROR_ACCESS_DENIED	The <i>hPolicy</i> handle was not opened with read/write access rights. The error is also returned if the client does not have the required credentials to call the method.
0x00000057 ERROR_INVALID_PARAMETER	One of the parameters of this method is incorrect, or is required and not specified. This error can be returned because: <ul style="list-style-type: none"> The <i>pCrypto</i> object did not pass the cryptographic set validations specified in the definition of the FW_CRYPTOSSET data type. One of the required values is not specified.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

3.1.4.24 RRPC_FWSetCryptoSet (Opnum 23)

The **RRPC_FWSetCryptoSet** method requests the server to modify the specified cryptographic set in the policy contained in the policy store referenced by the handle specified in the *hPolicy* parameter.

```
unsigned long RRPC_FWSetCryptoSet(
    [in] FW_CONN_HANDLE rpcConnHandle,
    [in] FW_POLICY_STORE_HANDLE hPolicy,
    [in] PFW_CRYPTOSSET pCrypto
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read/write access rights.

pCrypto: This parameter represents the cryptographic set the client wants to modify in the store. The set MUST be valid, as specified in the definition of the [FW_CRYPTOSSET](#) data type.

Return Values: The method returns 0 if successful; if failed, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000032 ERROR_NOT_SUPPORTED	The specified store does not support this method; the store might be read-only.
0x00000005 ERROR_ACCESS_DENIED	The <i>hPolicy</i> handle was not opened with read/write access rights. The error is also returned if the client does not have the required

Return value/code	Description
	credentials to call the method.
0x00000002 ERROR_FILE_NOT_FOUND	The specified set referenced by the wszSetID member string of the FW_CRYPTO_SET data type is not found in the policy store.
0x00000057 ERROR_INVALID_PARAMETER	One of the parameters of this method is incorrect, or is required and not specified. This error can be returned because: <ul style="list-style-type: none"> ▪ The <i>pCrypto</i> object did not pass the cryptographic set validations specified in the definition of the FW_CRYPTO_SET data type. ▪ One of the required values is not specified.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

This method modifies a cryptographic set in the cryptographic linked list of the memory representation of the store being modified. It also writes through and saves the set to the disk. If called on an online store, the method enumerates the connection security rules list and reapplies each rule referencing this default set to complete the enforcement of the policy.

When the server is operating under normal security mode, the server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method. When the server is operating under common criteria mode, the server MUST NOT allow any callers and return an access denied error.

3.1.4.25 RRPC_FWDeleteCryptoSet (Opnum 24)

The **RRPC_FWDeleteCryptoSet** method requests the server to delete the specified cryptographic set in the policy contained in the policy store that is referenced by the handle specified in the *hPolicy* parameter.

```
ULONG RRPC_FWDeleteCryptoSet(
    [in] FW_CONN_HANDLE rpcConnHandle,
    [in] FW_POLICY_STORE_HANDLE hPolicy,
    [in, range(FW_IPSEC_PHASE_INVALID+1, FW_IPSEC_PHASE_MAX-1)]
    FW_IPSEC_PHASE IpSecPhase,
    [in, string, ref] const wchar_t* wszSetId
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle that is successfully opened by using the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read/write access rights.

IpSecPhase: This parameter specifies the IPsec negotiation phase type in which this set is used.

wszSetId: This parameter is the pointer to a string that is the ID of the cryptographic set that the client wants to delete from the specified store.

Return Values: The method returns 0 if successful; if it fails, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000962 ERROR_ACTIVE_CONNECTIONS	The specified set is still referenced by connection security or main mode rules. This failure happens only when the set is not a default set. There is always a default set to use, either from other stores or a hard-coded one.
0x00000032 ERROR_NOT_SUPPORTED	The specified store does not support this method; the store might be read-only.
0x00000005 ERROR_ACCESS_DENIED	The <i>hPolicy</i> handle was not opened with read/write access rights. The error is also returned if the client does not have the required credentials to call the method.
0x00000002 ERROR_FILE_NOT_FOUND	The specified rule that is referenced by the wszSetID string is not found in the policy store.
0x00000057 ERROR_INVALID_PARAMETER	The specified IPsec phase is not a valid one.

Exceptions Thrown: No exceptions are thrown except those that are thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

This method deletes a cryptographic set in the cryptographic linked list of the memory representation of the store being modified. It also writes through and saves the set to disk. If called on an online store and the set is not a default set, the method does not delete the specified set if any connection rule references this set.

When the server is operating under normal security mode, the server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method. When the server is operating under common criteria mode, the server MUST NOT allow any callers and instead returns an access denied error.

3.1.4.26 RRPC_FWDeleteAllCryptoSets (Opnum 25)

The **RRPC_FWDeleteAllCryptoSets** method requests the server to delete all the cryptographic sets of a specific IPsec phase in the policy contained in the policy store that is referenced by the handle specified in the *hPolicy* parameter.

```
unsigned long RRPC_FWDeleteAllCryptoSets(
    [in] FW_CONN_HANDLE rpcConnHandle,
    [in] FW_POLICY_STORE_HANDLE hPolicy,
    [in, range(FW_IPSEC_PHASE_INVALID+1, FW_IPSEC_PHASE_MAX-1)]
    FW_IPSEC_PHASE IpSecPhase
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle that is successfully opened by using the

[RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read/write access rights.

IpSecPhase: This parameter specifies the IPsec negotiation phase type in which this set is used.

Return Values: The method returns 0 if successful; if it fails, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000962 ERROR_ACTIVE_CONNECTIONS	There are non-default sets still being referenced by connection security or main mode rules. There is always a default set to use, either from other stores or a hard-coded one; therefore, this failure never occurs because of default sets.
0x00000032 ERROR_NOT_SUPPORTED	The specified store does not support this method; the store might be read-only.
0x00000005 ERROR_ACCESS_DENIED	The <i>hPolicy</i> handle was not opened with read/write access rights. The error is also returned if the client does not have the required credentials to call the method.
0x00000057 ERROR_INVALID_PARAMETER	The specified IPsec phase is not a valid one.

Exceptions Thrown: No exceptions are thrown except those that are thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

This method deletes all the cryptographic sets in the cryptographic linked list of the memory representation of the store being modified. It also writes through and deletes the sets from disk. If called on an online store, the method does not delete the sets if any nondefault set is referenced by a connection security rule.

When the server is operating under normal security mode, the server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method. When the server is operating under common criteria mode, the server MUST NOT allow any callers and instead returns an access denied error.

3.1.4.27 RRPC_FWEnumCryptoSets (Opnum 26)

The **RRPC_FWEnumCryptoSets** method requests the server to return all the cryptographic sets of the specified IPsec phase contained in the store referenced by the *hPolicy* handle. The method returns a linked list of all these cryptographic objects.

```
unsigned long RRPC_FWEnumCryptoSets(  
    [in] FW_CONN_HANDLE rpcConnHandle,  
    [in] FW_POLICY_STORE_HANDLE hPolicy,  
    [in, range(FW_IPSEC_PHASE_INVALID+1, FW_IPSEC_PHASE_MAX-1)]  
        FW_IPSEC_PHASE IpSecPhase,  
    [in] unsigned long dwFilteredByStatus,  
    [in] unsigned short wFlags,  
    [out, ref] unsigned long* pdwNumSets,  
    [out] PFW_CRYPTOSSET* ppCryptoSets  
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read or read/write access rights.

IpSecPhase: This parameter specifies the specific IPsec negotiation phase to which this set applies.

dwFilteredByStatus: This parameter is a combination of flags from the [FW_RULE_STATUS_CLASS](#) enumeration. This method uses this bitmask to determine if rules should be returned or not. Sets that contain a status code of the class specified by matches to this parameter will be returned in the linked list.

wFlags: This parameter is a combination of flags from the [FW_ENUM_RULES_FLAGS](#) that modifies the behavior of the method and performs operations on the sets before returning them in the linked list.

pdwNumSets: This is an output parameter that on success MUST be equal to the number of sets returned.

ppCryptoSets: This is an output parameter that on success contains a linked list of [FW_CRYPTO_SET](#) data types.

Return Values: The method returns 0 if successful; if failed, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The client does not have the required credentials to call the method.
0x00000057 ERROR_INVALID_PARAMETER	One of the parameters of this method is incorrect, or is required and not specified. This error can be returned because: <ul style="list-style-type: none">▪ The <i>IpSecPhase</i> parameter specifies an invalid IPsec negotiation phase.▪ One of the required values is not specified.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

When this method is called, the server looks for the binary version of the client, which was associated with the hPolicy handle when the client sent the **RRPC_FWOpenPolicyStore()** call. The server compares this binary version parameter with the schema version that it supports. If the server has a schema version of 0x0201 and the client passed a 0x0200 binary version, the server removes all values that are not valid for a **FW_CRYPTO_SET** (section 2.2.66) structure that has a 0x0200 schema version. If the removed value was present on one or more suites of the set, the server removes those suites as a whole, leaving the remaining suites intact. For each set that had a

value removed, the server sets a FW_RULE_STATUS_PARTIALLY_IGNORED value on the **Status** field of the set. The client then receives cryptographic sets with values that correspond to the correct schema version, but the client recognizes that the information it has about the sets is potentially incomplete.

3.1.4.28 RRPC_FWEnumPhase1SAs (Opnum 27)

The **RRPC_FWEnumPhase1SAs** method requests the server to return all the security associations of the IPsec first negotiation phase contained in the store referenced by the *hPolicy* handle. The method returns a linked list of all these security associations.

```
unsigned long RRPC_FWEnumPhase1SAs (
    [in] FW_CONN_HANDLE rpcConnHandle,
    [in] FW_POLICY_STORE_HANDLE hPolicy,
    [in, unique] PFW_ENDPOINTS pEndpoints,
    [out, ref] unsigned long* pdwNumSAs,
    [out, size_is( , *pdwNumSAs)] PFW_PHASE1_SA_DETAILS* ppSAs
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. This handle MUST be of the FW_STORE_TYPE_DYNAMIC store.

pEndpoints: This parameter is a pointer to an [FW_ENDPOINTS](#) data type that can hold the addresses of the destination and source host. These addresses are used to match the security associations that will be returned. If this parameter is NULL, the method returns all IPsec first-phase security associations.

pdwNumSAs: This is an output parameter that on success MUST be equal to the number of security associations returned.

ppSAs: This is an output parameter that on success contains a linked list of [FW_PHASE1_SA_DETAILS](#) data types, each of which represents the first-phase security association.

Return Values: The method returns 0 if successful; if failed, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The client does not have the required credentials to call the method.
0x00000032 ERROR_NOT_SUPPORTED	The store handle is not of the dynamic store.
0x00000057 ERROR_INVALID_PARAMETER	One of the parameters of this method is incorrect, or is required and not specified.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.29 RRPC_FWEnumPhase2SAs (Opnum 28)

The **RRPC_FWEnumPhase2SAs** method requests the server to return all the security associations of the IPsec second negotiation phase contained in the store referenced by the *hPolicy* handle. The method returns a linked list of all these security associations.

```
unsigned long RRPC_FWEnumPhase2SAs (  
    [in] FW_CONN_HANDLE rpcConnHandle,  
    [in] FW_POLICY_STORE_HANDLE hPolicy,  
    [in, unique] PFW_ENDPOINTS pEndpoints,  
    [out, ref] unsigned long* pdwNumSAs,  
    [out, size_is(, *pdwNumSAs)] PFW_PHASE2_SA_DETAILS* ppSAs  
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. This handle MUST be of the FW_STORE_TYPE_DYNAMIC store.

pEndpoints: This parameter is a pointer to an [FW_ENDPOINTS](#) data type that can hold the addresses of the destination and source host. These addresses are used to match the security associations that will be returned. If this parameter is NULL, the method will return all IPsec second phase security associations. If an endpoint is empty (that is, equal to 0), the endpoint matches any address.

pdwNumSAs: This is an output parameter that on success MUST be equal to the number of security associations returned.

ppSAs: This is an output parameter that on success contains a linked list of [FW_PHASE2_SA_DETAILS](#) data types, each of which represents a second phase security association.

Return Values: The method returns 0 if successful; if failed, it returns a non-zero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following are common.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The client does not have the required credentials to call the method.
0x00000032 ERROR_NOT_SUPPORTED	The store handle is not of the dynamic store.
0x00000057 ERROR_INVALID_PARAMETER	One of the parameters of this method is incorrect, or is required and not specified.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.30 RRPC_FWDeletePhase1SAs (Opnum 29)

The **RRPC_FWDeletePhase1SAs** method requests the server to delete all the IPsec first negotiation phase security associations that match the specified endpoints.

```
unsigned long RRPC_FWDeletePhase1SAs(  
    [in] FW_CONN_HANDLE rpcConnHandle,  
    [in] FW_POLICY_STORE_HANDLE hPolicy,  
    [in, unique] PFW_ENDPOINTS pEndpoints  
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. This handle MUST be of the FW_STORE_TYPE_DYNAMIC store.

pEndpoints: This parameter is a pointer to an [FW_ENDPOINTS](#) data type that can hold the addresses of the destination and source host. These addresses are used to match the security associations that will be deleted. If this parameter is NULL, the method deletes all IPsec first-phase security associations. If an endpoint is empty (that is, equal to 0), the endpoint matches any address.

Return Values: The method returns 0 if successful; if failed, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The client does not have the required credentials to call the method.
0x00000032 ERROR_NOT_SUPPORTED	The store handle is not of the dynamic store.
0x00000057 ERROR_INVALID_PARAMETER	One of the parameters of this method is incorrect or is required and not specified.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.31 RRPC_FWDeletePhase2SAs (Opnum 30)

The **RRPC_FWDeletePhase2SAs (Opnum 30)** method requests the server to delete all the IPsec second-negotiation-phase security associations that match the specified endpoints.

```
unsigned long RRPC_FWDeletePhase2SAs(  
    [in] FW_CONN_HANDLE rpcConnHandle,  
    [in] FW_POLICY_STORE_HANDLE hPolicy,  
    [in, unique] PFW_ENDPOINTS pEndpoints  
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. This handle MUST be of the FW_STORE_TYPE_DYNAMIC store.

pEndpoints: This parameter is a pointer to an [FW_ENDPOINTS](#) data type that can hold the addresses of the destination and source host. These addresses are used to match the security associations that will be deleted. If this parameter is NULL, the method deletes all IPsec second-phase security associations. If an endpoint is empty (that is, equal to 0), the endpoint matches any address.

Return Values: The method returns 0 if successful; if failed, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The client does not have the required credentials to call the method.
0x00000032 ERROR_NOT_SUPPORTED	The store handle is not of the dynamic store.
0x00000057 ERROR_INVALID_PARAMETER	One of the parameters of this method is incorrect, or is required and not specified.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.32 RRPC_FWEnumProducts (Opnum 31)

The **RRPC_FWEnumProducts (Opnum 31)** method requests the server to return all the registered third-party software components registered with the firewall and advanced security component. The only method supported is binary version 0x020A.

```
unsigned long RRPC_FWEnumProducts(  
    [in] FW_CONN_HANDLE rpcConnHandle,  
    [in] FW_POLICY_STORE_HANDLE hPolicy,
```

```

[out] unsigned long* pdwNumProducts,
[out, size_is(*pdwNumProducts)]
    PFW_PRODUCT* ppProducts
);

```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. This handle MUST be of the [FW_STORE_TYPE_DYNAMIC](#) store.

pdwNumProducts: This is an output parameter that on success MUST be equal to the number of products returned.

ppProducts: An array of [FW_PRODUCT](#) data types, representing the registration of third-party software components.

Return Values: The method returns 0 if successful; if it fails, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The client does not have the required credentials to call the method.
0x00000032 ERROR_NOT_SUPPORTED	The store handle is not of the dynamic store.
0x00000057 ERROR_INVALID_PARAMETER	One of the parameters of this method either is incorrect or is required and not specified.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.33 RRPC_FWAddMainModeRule (Opnum 32)

The **RRPC_FWAddMainModeRule (Opnum 32)** method requests the server to add the main mode rule in the policy contained in the policy store referenced by the specified opened policy store handle. The only method supported is binary version 0x020A.

```

unsigned long RRPC_FWAddMainModeRule(
[in] FW_CONN_HANDLE rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE hPolicy,
[in] PFW_MM_RULE pMMRule,
[out] FW_RULE_STATUS* pStatus
);

```


rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC FWOpenPolicyStore \(Opnum 0\)](#) method. This handle MUST be of the [FW_STORE_TYPE_DYNAMIC](#) store.

pMMRule: This parameter represents the main mode rule that the client adds in the store. The rule MUST be valid, as specified in the definition of the [FW_MM_RULE](#) data type.

pStatus: This is an output parameter that on return will have the status code of the rule.

Return Values: The method returns 0 if successful; if failed, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The client does not have the required credentials to call the method.
0x000000B7 ERROR_ALREADY_EXISTS	The specified rule has a rule ID that already exists in the specified store.
0x00000057 ERROR_INVALID_PARAMETER	One of the parameters of this method either is incorrect or is required and not specified.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.34 RRPC_FWSetMainModeRule (Opnum 33)

The **RRPC_FWSetMainModeRule (Opnum 33)** method requests the server to modify the specified main mode rule in the policy contained in the policy store referenced by the handle specified in the *hPolicy* parameter. The only method supported is binary version 0x020A.

```
unsigned long RRPC_FWSetMainModeRule(  
    [in] FW_CONN_HANDLE rpcConnHandle,  
    [in] FW_POLICY_STORE_HANDLE hPolicy,  
    [in] PFW_MM_RULE pMMRule,  
    [out] FW_RULE_STATUS* pStatus  
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC FWOpenPolicyStore \(Opnum 0\)](#) method. This handle MUST be of the [FW_STORE_TYPE_DYNAMIC](#) store.

pMMRule: This parameter represents the main mode rule the client modifies in the store. The rule MUST be valid, as specified in the definition of the [FW_MM_RULE](#) data type.

pStatus: This is an output parameter that on return will have the status code of the rule.

Return Values: The method returns 0 if successful; if it fails, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The client does not have the required credentials to call the method.
0x00000002 ERROR_FILE_NOT_FOUND	The specified set referenced by the wszRuleID member STRING of the FW_MM_RULE data type is not found in the policy store.
0x00000057 ERROR_INVALID_PARAMETER	One of the parameters of this method either is incorrect or is required and not specified.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.35 RRPC_FWDeleteMainModeRule (Opnum 34)

The **RRPC_FWDeleteMainModeRule (Opnum 34)** method requests the server to delete the specified main mode rule in the policy contained in the policy store referenced by the handle specified in the *hPolicy* parameter. The only method supported is binary version 0x020A.

```
unsigned long RRPC_FWDeleteMainModeRule (  
    [in] FW_CONN_HANDLE rpcConnHandle,  
    [in] FW_POLICY_STORE_HANDLE hPolicy,  
    [in, string, ref] LPCWSTR pRuleId  
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. This handle MUST be of the [FW_STORE_TYPE_DYNAMIC](#) store.

pRuleId: This parameter is the pointer to a [STRING](#) that is the ID of the main mode rule the client deletes from the specified store.

Return Values: The method returns 0 if successful; if it fails, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The client does not have the required credentials to call the method.
0x00000002 ERROR_FILE_NOT_FOUND	The specified set referenced by the wszRuleID member string of the FW_MM_RULE data type is not found in the policy store.
0x00000057 ERROR_INVALID_PARAMETER	One of the parameters of this method either is incorrect or is required and not specified.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.36 RRPC_FWDeleteAllMainModeRules (Opnum 35)

The **RRPC_FWDeleteAllMainModeRules (Opnum 35)** method requests the server to delete all the main mode rules in the policy contained in the policy store referenced by the handle specified in the *hPolicy* parameter. The only method supported is binary version 0x020A.

```
unsigned long RRPC_FWDeleteAllMainModeRules(
    [in] FW_CONN_HANDLE rpcConnHandle,
    [in] FW_POLICY_STORE_HANDLE hPolicy
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. This handle MUST be of the [FW_STORE_TYPE_DYNAMIC](#) store.

Return Values: The method returns 0 if successful; if it fails, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000032 ERROR_NOT_SUPPORTED	The specified store does not support this method; the store might be read-only.
0x00000005 ERROR_ACCESS_DENIED	The <i>hPolicy</i> handle was not opened with read/write access rights. The error is also returned if the client does not have the required credentials to call the method.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in [3.1.4](#)) before executing this method.

3.1.4.37 RRPC_FWEnumMainModeRules (Opnum 36)

The **RRPC_FWEnumMainModeRules (Opnum 36)** method requests the server to return all the main mode rules contained in the store referenced by the *hPolicy* handle. The method returns a linked list of all the main mode rule objects. The only method supported is binary version 0x020A.

```
unsigned long RRPC_FWEnumMainModeRules(  
    [in] FW_CONN_HANDLE rpcConnHandle,  
    [in] FW_POLICY_STORE_HANDLE hPolicy,  
    [in] unsigned long dwFilteredByStatus,  
    [in] unsigned long dwProfileFilter,  
    [in] unsigned short wFlags,  
    [out, ref] unsigned long* pdwNumRules,  
    [out] PFW_MM_RULE* ppMMRules  
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. This handle MUST be of the [FW_STORE_TYPE_DYNAMIC](#) store.

dwFilteredByStatus: This parameter is a combination of flags from the [FW_RULE_STATUS_CLASS](#) enumeration. This method uses this bitmask to determine whether rules should be returned. Rules that contain a status code of the class specified by this parameter will be returned in the linked list.

dwProfileFilter: This parameter is a combination of flags from the [FW_PROFILE_TYPE](#) enumeration. This method also uses this parameter to determine whether rules should be returned. Rules that contain a profile specified by this parameter will be returned in the linked list.

wFlags: This parameter is a combination of flags from the [FW_ENUM_RULES_FLAGS](#) enumeration, which modifies the behavior of the method and performs operations on the rules before returning them in the linked list.

pdwNumRules: This is an output parameter that on success MUST be equal to the number of rules returned.

ppMMRules: This is an output parameter that on success contains a linked list of [FW_MM_RULE](#) data types.

Return Values: The method returns 0 if successful; if it fails, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The client does not have the required credentials to call the method.
0x00000057 ERROR_INVALID_PARAMETER	The <i>dwProfileFilter</i> parameter contains invalid profiles.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.38 RRPC_FWQueryFirewallRules (Opnum 37)

The **RRPC_FWQueryFirewallRules (Opnum 37)** method requests the server to return all the firewall rules that match the specified query object that are contained in the store referenced by the *hPolicy* handle. The method returns a linked list of all the firewall rule objects. The only method supported is binary version 0x020A.

```
unsigned long RRPC_FWQueryFirewallRules(  
    [in] FW_CONN_HANDLE rpcConnHandle,  
    [in] FW_POLICY_STORE_HANDLE hPolicy,  
    [in] PFW_QUERY pQuery,  
    [in] unsigned short wFlags,  
    [out, ref] unsigned long* pdwNumRules,  
    [out] PFW_RULE2_0* ppRule  
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. This handle MUST be of the [FW_STORE_TYPE_DYNAMIC](#) store.

pQuery: This parameter represents the query object that the client uses to specify which main mode rules MUST be retrieved from the store. The query object MUST be valid, as specified in the definition of the [FW_QUERY](#) data type.

wFlags: This parameter is a combination of flags from the [FW_ENUM_RULES_FLAGS](#) enumeration, which modifies the behavior of the method and performs operations on the rules before returning them in the linked list.

pdwNumRules: This is an output parameter that on success MUST be equal to the number of rules returned.

ppRule: This is an output parameter that on success contains a linked list of [FW_RULE2_0](#) data types.

Return Values: The method returns 0 if successful; if it fails, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The client does not have the required credentials to call the method.
0x00000057 ERROR_INVALID_PARAMETER	The <i>pQuery</i> parameter contains invalid profiles.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.39 RRPC_FWQueryConnectionSecurityRules (Opnum 38)

The **RRPC_FWQueryConnectionSecurityRules (Opnum 38)** method requests the server to return all the connection security rules that match the specified query object that are contained in the store referenced by the *hPolicy* handle. The method returns a linked list of all the connection security rule objects. The only method supported is binary version 0x020A.

```
unsigned long RRPC_FWQueryConnectionSecurityRules(  
    [in] FW_CONN_HANDLE rpcConnHandle,  
    [in] FW_POLICY_STORE_HANDLE hPolicy,  
    [in] PFW_QUERY pQuery,  
    [in] unsigned short wFlags,  
    [out, ref] unsigned long* pdwNumRules,  
    [out] PFW_CS_RULE* ppRules  
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. This handle MUST be of the [FW_STORE_TYPE_DYNAMIC](#) store.

pQuery: This parameter represents the query object that the client uses to specify which main mode rules MUST be retrieved from the store. The query object MUST be valid, as specified in the definition of the [FW_QUERY](#) data type.

wFlags: This parameter is a combination of flags from the [FW_ENUM_RULES_FLAGS](#), which modifies the behavior of the method and performs operations on the rules before returning them in the linked list.

pdwNumRules: This is an output parameter that on success MUST be equal to the number of rules returned.

ppRules: This is an output parameter that on success contains a linked list of [FW_CS_RULE](#) data types.

Return Values: The method returns 0 if successful; if failed, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The client does not have the required credentials to call the method.
0x00000057 ERROR_INVALID_PARAMETER	The <i>pQuery</i> parameter contains invalid profiles.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.40 RRPC_FWQueryMainModeRules (Opnum 39)

The **RRPC_FWQueryMainModeRules (Opnum 39)** method requests the server to return all the main mode rules that match the specified query object that are contained in the store referenced by the *hPolicy* handle. The method returns a linked list of all the main mode rule objects. The only method supported is binary version 0x020A.

```
unsigned long RRPC_FWQueryMainModeRules(  
    [in] FW_CONN_HANDLE rpcConnHandle,  
    [in] FW_POLICY_STORE_HANDLE hPolicy,  
    [in] PFW_QUERY pQuery,  
    [in] unsigned short wFlags,  
    [out, ref] unsigned long* pdwNumRules,  
    [out] PFW_MM_RULE ppMMRules  
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. This handle MUST be of the [FW_STORE_TYPE_DYNAMIC](#) store.

pQuery: This parameter represents the query object that the client uses to specify which main mode rules MUST be retrieved from the store. The query object MUST be valid, as specified in the definition of the [FW_QUERY](#) data type.

wFlags: This parameter is a combination of flags from the [FW_ENUM_RULES_FLAGS](#) enumeration, which modifies the behavior of the method and performs operations on the rules before returning them in the linked list.

pdwNumRules: This is an output parameter that on success MUST be equal to the number of rules returned.

ppMMRules: This is an output parameter that on success contains a linked list of [FW_MM_RULE](#) data types.

Return Values: The method returns 0 if successful; if it fails, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The client does not have the required credentials to call the method.
0x00000057 ERROR_INVALID_PARAMETER	The <i>pQuery</i> parameter contains invalid profiles.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.41 RRPC_FWQueryAuthenticationSets (Opnum 40)

The **RRPC_FWQueryAuthenticationSets (Opnum 40)** method requests the server to return all the authentication sets that match the specified query object that are contained in the store referenced by the *hPolicy* handle. The method returns a linked list of all the authentication set objects. The only method supported is binary version 0x020A.

```
unsigned long RRPC_FWQueryAuthenticationSets(  
    [in] FW_CONN_HANDLE rpcConnHandle,  
    [in] FW_POLICY_STORE_HANDLE hPolicy,  
    [in, range(FW_IPSEC_PHASE_INVALID+1, FW_IPSEC_PHASE_MAX-1)]  
        FW_IPSEC_PHASE IPsecPhase,  
    [in] PFW_QUERY pQuery,  
    [in] unsigned short wFlags,  
    [out, ref] unsigned long* pdwNumSets,  
    [out] PFW_AUTH_SET* ppAuthSets  
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW POLICY STORE HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC FWOpenPolicyStore \(Opnum 0\)](#) method. This handle MUST be of the [FW STORE TYPE DYNAMIC](#) store.

IPsecPhase: This parameter specifies the specific IPsec negotiation phase to which this set applies.

pQuery: This parameter represents the query object that the client wants to use to specify which main mode rules MUST be retrieved from the store. The query object MUST be valid, as specified in the definition of the [FW QUERY](#) data type.

wFlags: This parameter is a combination of flags from the [FW ENUM RULES FLAGS](#) enumeration, which modifies the behavior of the method and performs operations on the rules before returning them in the linked list.

pdwNumSets: This is an output parameter that, on success, MUST be equal to the number of sets returned.

ppAuthSets: This is an output parameter that on success contains a linked list of [FW AUTH SET](#) data types.

Return Values: The method returns 0 if successful; if it fails, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000005	The client does not have the required credentials to call the

Return value/code	Description
ERROR_ACCESS_DENIED	method.
0x00000057 ERROR_INVALID_PARAMETER	The <i>dwProfileFilter</i> parameter contains invalid profiles.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.42 RRPC_FWQueryCryptoSets (Opnum 41)

The **RRPC_FWQueryCryptoSets (Opnum 41)** method requests the server to return all the crypto sets that match the specified query object that are contained in the store referenced by the **hPolicy** handle. The method returns a linked list of all the crypto set objects. The only method supported is binary version 0x020A.

```

unsigned long RRPC_FWQueryCryptoSets(
    [in] FW_CONN_HANDLE rpcConnHandle,
    [in] FW_POLICY_STORE_HANDLE hPolicy,
    [in, range(FW_IPSEC_PHASE_INVALID+1, FW_IPSEC_PHASE_MAX-1)]
        FW_IPSEC_PHASE IPsecPhase,
    [in] PFW_QUERY pQuery,
    [in] unsigned short wFlags,
    [out, ref] unsigned long* pdwNumSets,
    [out] PFW_CRYPTO_SET* ppCryptoSets
);

```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. This handle MUST be of the [FW_STORE_TYPE_DYNAMIC](#) store.

IPsecPhase: This parameter specifies the specific IPsec negotiation phase to which this set applies.

pQuery: This parameter represents the query object that the client wants to use to specify which main mode rules MUST be retrieved from the store. The query object MUST be valid, as specified in the definition of the [FW_QUERY](#) data type.

wFlags: This parameter is a combination of flags from the [FW_ENUM_RULES_FLAGS](#) enumeration, which modifies the behavior of the method and performs operations on the rules before returning them in the linked list.

pdwNumSets: This is an output parameter that, on success, MUST be equal to the number of sets returned.

ppCryptoSets: This is an output parameter that, on success, contains a linked list of [FW_CRYPTO_SET](#) data types.

Return Values: The method returns 0 if successful; if it fails, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The client does not have the required credentials to call the method.
0x00000057 ERROR_INVALID_PARAMETER	The <i>pQuery</i> parameter contains invalid profiles.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.43 RRPC_FWEnumNetworks (Opnum 42)

The **RRPC_FWEnumNetworks (Opnum 42)** method requests the server to return all the networks to which the host with the firewall and advanced security component is connected. The only method supported is binary version 0x020A.

```

unsigned long RRPC_FWEnumNetworks(
    [in] FW_CONN_HANDLE rpcConnHandle,
    [in] FW_POLICY_STORE_HANDLE hPolicy,
    [out] unsigned long pdwNumNetworks,
    [out, size_is(*pdwNumNetworks)]
    PFW_NETWORK* ppNetworks
);

```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. This handle MUST be of the [FW_STORE_TYPE_DYNAMIC](#) store.

pdwNumNetworks: This is an output parameter that, on success, MUST be equal to the number of networks returned.

ppNetworks: This is an output parameter that, on success, contains an array of [FW_NETWORK](#) data types.

Return Values: The method returns 0 if successful; if it fails, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The client does not have the required credentials to call the method.

Return value/code	Description
0x00000057 ERROR_INVALID_PARAMETER	A parameter contains invalid profiles.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.44 RRPC_FWEnumAdapters (Opnum 43)

The **RRPC_FWEnumAdapters (Opnum 43)** method requests the server to return all the networks interfaces that the host with the firewall and advanced security component has. The only method supported is binary version 0x020A.

```

unsigned long RRPC_FWEnumAdapters(
    [in] FW_CONN_HANDLE rpcConnHandle,
    [in] FW_POLICY_STORE_HANDLE hPolicy,
    [out] unsigned long pdwNumAdapters,
    [out, size_is(*pdwNumAdapters)]
    PFW_ADAPTER* ppAdapters
);

```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. This handle MUST be of the [FW_STORE_TYPE_DYNAMIC](#) store.

pdwNumAdapters: This is an output parameter that, on success, MUST be equal to the number of networks returned.

ppAdapters: This is an output parameter that, on success, contains an array of [FW_ADAPTER](#) data types.

Return Values: The method returns 0 if successful; if it fails, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The client does not have the required credentials to call the method.
0x00000057 ERROR_INVALID_PARAMETER	A parameter contains invalid profiles.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.45 RRPC_FWGetGlobalConfig2_10 (Opnum 44)

The **RRPC_FWGetGlobalConfig2_10 (Opnum 44)** method retrieves the value of a global policy configuration option. The client specifies to the server from which store this value MUST be retrieved and in which specific configuration option it is interested. The only method supported is binary version 0x020A.

```
unsigned long RRPC_FWGetGlobalConfig2_10(  
    [in] FW_CONN_HANDLE rpcConnHandle,  
    [in] unsigned short BinaryVersion,  
    [in] FW_STORE_TYPE StoreType,  
    [in, range(FW_GLOBAL_CONFIG_INVALID+1, FW_GLOBAL_CONFIG_MAX-1)]  
        FW_GLOBAL_CONFIG configID,  
    [in] unsigned long dwFlags,  
    [in, out, unique, size_is(cbData), length_is(*pcbTransmittedLen)]  
        BYTE* pBuffer,  
    [in] unsigned long cbData,  
    [in, out] unsigned long* pcbTransmittedLen,  
    [out] unsigned long* pcbRequired,  
    [out] FW_RULE_ORIGIN_TYPE* pOrigin  
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

BinaryVersion: This parameter specifies the RPC interface binary version. This implies versions of the methods and versions of the structures.

StoreType: This parameter specifies the policy store from which the client retrieves the configuration option value.

configID: This parameter specifies the specific global policy configuration option that the client is interested in retrieving.

dwFlags: This parameter is a combination of flags from the [FW_CONFIG_FLAGS](#) enumeration, which modifies the behavior of this method, as specified in the definition of the enumeration.

pBuffer: This is an input/output parameter. This parameter is a pointer to the buffer that the client provides to contain the value of the profile configuration option that is being requested.

cbData: This parameter is the size of the buffer to which the *pBuffer* parameter points.

pcbTransmittedLen: This is a pointer to an input and output parameter that specifies the length of the transmitted data within the buffer.

pcbRequired: This is a pointer to an output parameter that specifies the required minimum buffer size, in octets, for the method to be able to return the configuration value. This output parameter is nonzero only if the buffer (pointed to by *pBuffer* and whose size is *cbData*) was not big enough to contain the value.

pOrigin: This field is the origin of the configuration option, as specified in the [FW_RULE_ORIGIN_TYPE](#) enumeration. On success, it MUST be filled.

Return Values: The method returns 0 if successful; if it fails, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The client does not have the required credentials to call the method.
0x00000002 ERROR_FILE_NOT_FOUND	The specific configuration option is not found within the policy. This means that it is not configured. If the option is not configured in any other store, the firewall uses a default value.
0x00000032 ERROR_NOT_SUPPORTED	The specified store type does not support this method.
0x00000057 ERROR_INVALID_PARAMETER	One of the parameters of this method either is incorrect or is required and not specified. This error can be returned because: <ul style="list-style-type: none"> ▪ The specific configuration option is not meant to be available in the specified store. ▪ The specified configuration option is not defined. ▪ One of the required values is not specified. ▪ The buffer is not big enough to hold the specific value.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.46 RRPC_FWGetConfig2_10 (Opnum 45)

The **RRPC_FWGetConfig2_10 (Opnum 45)** method retrieves the value of a profile configuration option. The client specifies to the server from which store and profile this value MUST be retrieved and in which specific configuration option it is interested. The only method supported is binary version 0x020A.

```

unsigned long RRPC_FWGetConfig2_10(
    [in] FW_CONN_HANDLE rpcConnHandle,
    [in] FW_POLICY_STORE_HANDLE hPolicyStore,
    [in, range(FW_GLOBAL_CONFIG_INVALID+1, FW_GLOBAL_CONFIG_MAX-1)]
        FW_GLOBAL_CONFIG configID,
    [in] FW_PROFILE_TYPE Profile,
    [in] unsigned long dwFlags,
    [in, out, unique, size_is(cbData), length_is(*pcbTransmittedLen)]
        BYTE* pBuffer,
    [in] unsigned long cbData,
    [in, out] unsigned long* pcbTransmittedLen,
    [out] unsigned long* pcbRequired,
    [out] FW_RULE_ORIGIN_TYPE* pOrigin
);

```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicyStore: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read/write access rights.

configID: This parameter specifies the specific global policy configuration option that the client is interested in retrieving.

Profile: This parameter specifies from which specific profile this value MUST be retrieved.

dwFlags: This parameter is a combination of flags from the [FW_CONFIG_FLAGS](#) enumeration, which modifies the behavior of this method, as specified in the definition of the enumeration.

pBuffer: This is an input/output parameter. This parameter is a pointer to the buffer that the client provides to contain the value of the profile configuration option being requested.

cbData: This parameter is the size of the buffer to which the *pBuffer* parameter points.

pcbTransmittedLen: This is a pointer to an input and output parameter that specifies the length of the transmitted data within the buffer.

pcbRequired: This is a pointer to an output parameter that specifies the required minimum buffer size, in octets, for the method to be able to return the configuration value. This output parameter is nonzero only if the buffer (pointed to by *pBuffer* and whose size is *cbData*) was not big enough to contain the value.

pOrigin: This field is the origin of the configuration option, as specified in the [FW_RULE_ORIGIN_TYPE](#) enumeration. On success, it MUST be filled.

Return Values: The method returns 0 if successful; if it fails, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The client does not have the required credentials to call the method.
0x00000002 ERROR_FILE_NOT_FOUND	The specific configuration option is not found within the policy. This means that it is not configured. If the option is not configured in any other store, the firewall uses a default value.
0x00000032 ERROR_NOT_SUPPORTED	The store type specified does not support this method.
0x000000EA ERROR_MORE_DATA	The buffer is not big enough to hold the configuration option value.
0x00000057 ERROR_INVALID_PARAMETER	One of the parameters of this method either is incorrect or is required and not specified. This error can be returned because: <ul style="list-style-type: none">▪ The specific configuration option is not meant to be available in the specified store.

Return value/code	Description
	<ul style="list-style-type: none"> ▪ The specified configuration option is not defined. ▪ One of the required values is not specified. ▪ The buffer is not big enough to hold the specific value.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.47 RRPC_FWAddFirewallRule2_10 (Opnum 46)

The **RRPC_FWAddFirewallRule2_10 (Opnum 46)** method requests the server to add the specified firewall rule in the policy contained in the policy store referenced by the handle specified in the *hPolicyStore* parameter. The only method supported is binary version 0x020A.

```

unsigned long RRPC_FWAddFirewallRule2_10(
    [in] FW_CONN_HANDLE rpcConnHandle,
    [in] FW_POLICY_STORE_HANDLE hPolicyStore,
    [in] PFW_RULE pRule,
    [out] FW_RULE_STATUS* pStatus
);

```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicyStore: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read/write access rights.

pRule: This parameter represents the firewall rule that the client wants to add to the store. The rule MUST be a valid rule, as specified in the definition of the [FW_RULE](#) data type.

pStatus: This output parameter is the status code of the rule as specified by the [FW_RULE_STATUS](#) enumeration. This field is filled out on return.

Return Values: The method returns 0 if successful; if it fails, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x000000B7 ERROR_ALREADY_EXISTS	The specified rule has a rule ID that already exists in the specified store.
0x00000032 ERROR_NOT_SUPPORTED	The specified store does not support this method; the store might be read-only.
0x00000005 ERROR_ACCESS_DENIED	The <i>hPolicyStore</i> handle was not opened with read/write access rights. The error is also returned if the client does not have the

Return value/code	Description
	required credentials to call the method.
0x00000057 ERROR_INVALID_PARAMETER	<p>One of the parameters of this method either is incorrect or is required and not specified. This error can be returned because:</p> <ul style="list-style-type: none"> ▪ The <i>pRule</i> object did not pass the firewall rule validations specified in the definition of the FW_RULE data type. ▪ One of the required values is not specified. ▪ A policy store does not support rules with profile conditions other than ALL profiles. ▪ The <i>wszLocalApplication</i> parameter contains a string that at enforcement time does not represent a valid file path. <18>

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

This method adds a firewall rule in the firewall linked list of the memory representation of the store being modified. It also writes through and saves the rule on disk. If called on an online store, the firewall rule is also enforced.

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.48 RRPC_FWSetFirewallRule2_10 (Opnum 47)

The **RRPC_FWSetFirewallRule2_10 (Opnum 47)** method requests the server to modify the specified firewall rule in the policy contained in the policy store referenced by the handle specified in the *hPolicyStore* parameter. The only method supported is binary version 0x020A.

```

unsigned long RRPC_FWSetFirewallRule2_10(
    [in] FW_CONN_HANDLE rpcConnHandle,
    [in] FW_POLICY_STORE_HANDLE hPolicyStore,
    [in] PFW_RULE pRule,
    [out] FW_RULE_STATUS* pStatus
);

```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicyStore: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read/write access rights.

pRule: This parameter represents the firewall rule that the client wants to add to the store. The rule MUST be a valid rule, as specified in the definition of the [FW_RULE](#) data type.

pStatus: This output parameter is the status code of the rule as specified by the [FW_RULE_STATUS](#) enumeration. This field is filled out on return.

Return Values: The method returns 0 if successful; if it fails, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000032 ERROR_NOT_SUPPORTED	The specified store does not support this method; the store might be read-only.
0x00000005 ERROR_ACCESS_DENIED	The <i>hPolicyStore</i> handle was not opened with read/write access rights. The error is also returned if the client does not have the required credentials to call the method.
0x00000002 ERROR_FILE_NOT_FOUND	The specified rule referenced by the <i>wszRuleID</i> member string of the FW_RULE data type is not found in the policy store.
0x00000057 ERROR_INVALID_PARAMETER	One of the parameters of this method either is incorrect or is required and not specified. This error can be returned because: <ul style="list-style-type: none"> ▪ The <i>pRule</i> object did not pass the firewall rule validations specified in the definition of the FW_RULE data type. ▪ One of the required values is not specified. ▪ A policy store does not support rules with profile conditions other than ALL profiles.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.49 RRPC_FWEnumFirewallRules2_10 (Opnum 48)

The **RRPC_FWEnumFirewallRules2_10 (Opnum 48)** method requests the server to return all the firewall rules contained in the store referenced by the *hPolicyStore* handle. The method returns a linked list of all the firewall rule objects. The only method supported is binary version 0x020A.

```

unsigned long RRPC_FWEnumFirewallRules2_10 (
    [in] FW_CONN_HANDLE rpcConnHandle,
    [in] FW_POLICY_STORE_HANDLE hPolicyStore,
    [in] unsigned long dwFilteredByStatus,
    [in] unsigned long dwProfileFilter,
    [in] unsigned short wFlags,
    [out, ref] unsigned long* pdwNumRules,
    [out] PFW_RULE* ppRules
);

```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicyStore: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read/write access rights.

dwFilteredByStatus: This parameter is a combination of flags from the [FW_RULE_STATUS_CLASS](#) enumeration. This method uses this bitmask to determine whether rules should be returned. Rules that contain a status code of the specified class that match this parameter will be returned in the linked list.

dwProfileFilter: This parameter is a combination of flags from the [FW_PROFILE_TYPE](#) enumeration. This method also uses this parameter to determine whether rules should be returned. Rules that contain a profile specified by this parameter will be returned in the linked list.

wFlags: This parameter is a combination of flags from the [FW_ENUM_RULES_FLAGS](#) enumeration, which modifies the behavior of the method and performs operations on the rules before returning them in the linked list.

pdwNumRules: This is an output parameter that, on success, MUST be equal to the number of rules returned.

ppRules: This is an output parameter that, on success, contains a linked list of [FW_RULE](#) data types.

Return Values: The method returns 0 if successful; if it fails, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The <i>hPolicyStore</i> handle was not opened with read/write access rights. The error is also returned if the client does not have the required credentials to call the method.
0x00000057 ERROR_INVALID_PARAMETER	The <i>dwProfileFilter</i> parameter contains invalid profiles.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.50 RRPC_FWAddConnectionSecurityRule2_10 (Opnum 49)

The **RRPC_FWAddConnectionSecurityRule2_10 (Opnum 49)** method requests the server to add the specified connection security rule in the policy contained in the policy store referenced by the handle specified in the *hPolicyStore* parameter. The only method supported is binary version 0x020A.

```
unsigned long RRPC_FWAddConnectionSecurityRule2_10(  
    [in] FW_CONN_HANDLE rpcConnHandle,  
    [in] FW_POLICY_STORE_HANDLE hPolicyStore,  
    [in] PFW_CS_RULE pRule,  
    [out] FW_RULE_STATUS* pStatus  
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicyStore: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read/write access rights.

pRule: This parameter represents the firewall rule that the client adds to the store. The rule MUST be a valid rule, as specified in the definition of the [FW_CS_RULE](#) data type.

pStatus: This output parameter is the status code of the rule as specified by the [FW_RULE_STATUS](#) enumeration. This field is filled out on return.

Return Values: The method returns 0 if successful; if it fails, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x000000B7 ERROR_ALREADY_EXISTS	The specified rule has a rule ID that already exists in the specified store.
0x00000032 ERROR_NOT_SUPPORTED	The specified store does not support this method; the store might be read-only.
0x00000005 ERROR_ACCESS_DENIED	The <i>hPolicyStore</i> handle was not opened with read/write access rights. The error is also returned if the client does not have the required credentials to call the method.
0x00000057 ERROR_INVALID_PARAMETER	One of the parameters of this method either is incorrect or is required and not specified. This error can be returned because: <ul style="list-style-type: none"> ▪ The <i>pRule</i> object did not pass the firewall rule validations specified in the definition of the FW_CS_RULE data type. ▪ One of the required values is not specified. ▪ A policy store does not support rules with profile conditions other than ALL profiles.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

This method adds a firewall rule in the firewall linked list of the memory representation of the store being modified. It also writes through and saves the rule on disk. If called on an online store, the firewall rule is also enforced.

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.51 RRPC_FWSetConnectionSecurityRule2_10 (Opnum 50)

The **RRPC_FWSetConnectionSecurityRule2_10 (Opnum 50)** method requests the server to modify the specified connection security rule in the policy contained in the policy store referenced by the handle specified in the *hPolicyStore* parameter. The only method supported is binary version 0x020A.

```
unsigned long RRPC_FWSetConnectionSecurityRule2_10(
```

```

[in] FW_CONN_HANDLE rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE hPolicyStore,
[in] PFW_CS_RULE pRule,
[out] FW_RULE_STATUS* pStatus
);

```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicyStore: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read/write access rights.

pRule: This parameter represents the connection security rule that the client wants to add to the store. The rule MUST be a valid rule, as specified in the definition of the [FW_CS_RULE](#) data type.

pStatus: This output parameter is the status code of the rule as specified by the [FW_RULE_STATUS](#) enumeration. This field is filled out on return.

Return Values: The method returns 0 if successful; if it fails, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000032 ERROR_NOT_SUPPORTED	The specified store does not support this method; the store might be read-only.
0x00000005 ERROR_ACCESS_DENIED	The <i>hPolicyStore</i> handle was not opened with read/write access rights. The error is also returned if the client does not have the required credentials to call the method.
0x00000002 ERROR_FILE_NOT_FOUND	The specified rule referenced by the wszRuleID member string of the FW_CS_RULE data type is not found in the policy store.
0x00000057 ERROR_INVALID_PARAMETER	One of the parameters of this method either is incorrect or is required and not specified. This error can be returned because: <ul style="list-style-type: none"> ▪ The <i>pRule</i> object did not pass the firewall rule validations specified in the definition of the FW_CS_RULE data type. ▪ One of the required values is not specified. ▪ A policy store does not support rules with profile conditions other than ALL profiles.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.52 RRPC_FWEnumConnectionSecurityRules2_10 (Opnum 51)

The **RRPC_FWEnumConnectionSecurityRules2_10 (Opnum 51)** method requests the server to return all the connection security rules contained in the store referenced by the *hPolicyStore* handle. The method returns a linked list of all the connection security rule objects. The only method supported is binary version 0x020A.

```
unsigned long RRPC_FWEnumConnectionSecurityRules2_10(  
    [in] FW_CONN_HANDLE rpcConnHandle,  
    [in] FW_POLICY_STORE_HANDLE hPolicyStore,  
    [in] unsigned long dwFilteredByStatus,  
    [in] unsigned long dwProfileFilter,  
    [in] unsigned short wFlags,  
    [out, ref] unsigned long* pdwNumRules,  
    [out] PFW_CS_RULE* ppRules  
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicyStore: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read/write access rights.

dwFilteredByStatus: This parameter is a combination of flags from the [FW_RULE_STATUS_CLASS](#) enumeration. This method uses this bitmask to determine whether rules should be returned. Rules that contain a status code of the specified class that match this parameter will be returned in the linked list.

dwProfileFilter: This parameter is a combination of flags from the [FW_PROFILE_TYPE](#) enumeration. This method also uses this parameter to determine whether rules should be returned. Rules that contain a profile specified by this parameter will be returned in the linked list.

wFlags: This parameter is a combination of flags from the [FW_ENUM_RULES_FLAGS](#) enumeration, which modifies the behavior of the method and performs operations on the rules before returning them in the linked list.

pdwNumRules: This is an output parameter that on success MUST be equal to the number of rules returned.

ppRules: This is an output parameter that on success contains a linked list of [FW_CS_RULE](#) data types.

Return Values: The method returns 0 if successful; if it fails, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The <i>hPolicyStore</i> handle was not opened with read/write access rights. The error is also returned if the client does not have the required credentials to call the method.

Return value/code	Description
0x00000057 ERROR_INVALID_PARAMETER	The <i>dwProfileFilter</i> parameter contains invalid profiles.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.53 RRPC_FWAddAuthenticationSet2_10 (Opnum 52)

The **RRPC_FWAddAuthenticationSet2_10 (Opnum 52)** method requests the server to add the authentication set in the policy contained in the policy store referenced by the handle specified in the *hPolicy* parameter. The only method supported is binary version 0x020A.

```
unsigned long RRPC_FWAddAuthenticationSet2_10(
    [in] FW_CONN_HANDLE rpcConnHandle,
    [in] FW_POLICY_STORE_HANDLE hPolicy,
    [in] PFW_AUTH_SET pAuth,
    [out] FW_RULE_STATUS* pStatus
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read/write access rights.

pAuth: This parameter represents the authentication set that the client wants to add to the store. The set MUST be valid, as specified in the definition of the [FW_AUTH_SET](#) data type.

pStatus: This output parameter is the status code of the rule as specified by the [FW_RULE_STATUS](#) enumeration. This field is filled out on return.

Return Values: The method returns 0 if successful; if it fails, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x000000B7 ERROR_ALREADY_EXISTS	The specified rule has a rule ID that already exists in the specified store.
0x00000032 ERROR_NOT_SUPPORTED	The specified store does not support this method; the store might be read-only.
0x00000005 ERROR_ACCESS_DENIED	The <i>hPolicy</i> handle was not opened with read/write access rights. The error is also returned if the client does not have the required credentials to call the method.
0x00000057	One of the parameters of this method either is incorrect or is

Return value/code	Description
ERROR_INVALID_PARAMETER	required and not specified. This error can be returned because: <ul style="list-style-type: none"> The <i>pAuth</i> object did not pass the firewall rule validations specified in the definition of the FW_AUTH_SET data type. One of the required values is not specified.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

This method adds a firewall rule in the firewall linked list of the memory representation of the store being modified. It also writes through and saves the rule on disk. If the method is called on an online store, the firewall rule is also enforced.

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.54 RRPC_FWSetAuthenticationSet2_10 (Opnum 53)

The **RRPC_FWSetAuthenticationSet2_10 (Opnum 53)** method requests the server to modify the specified authentication set in the policy contained in the policy store referenced by the handle specified in the *hPolicy* parameter. The only method supported is binary version 0x020A.

```
unsigned long RRPC_FWSetAuthenticationSet2_10(
    [in] FW_CONN_HANDLE rpcConnHandle,
    [in] FW_POLICY_STORE_HANDLE hPolicy,
    [in] PFW_AUTH_SET pAuth,
    [out] FW_RULE_STATUS* pStatus
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read/write access rights.

pAuth: This parameter represents the authentication set that the client wants to add to the store. The set MUST be valid, as specified in the definition of the [FW_AUTH_SET](#) data type.

pStatus: This output parameter is the status code of the rule as specified by the [FW_RULE_STATUS](#) enumeration. This field is filled out on return.

Return Values: The method returns 0 if successful; if it fails, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000032 ERROR_NOT_SUPPORTED	The specified store does not support this method; the store might be read-only.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The <i>hPolicy</i> handle was not opened with read/write access rights. The error is also returned if the client does not have the required credentials to call the method.
0x00000002 ERROR_FILE_NOT_FOUND	The specified rule referenced by the wszSetID member string of the FW_AUTH_SET data type is not found in the policy store.
0x00000057 ERROR_INVALID_PARAMETER	One of the parameters of this method either is incorrect or is required and not specified. This error can be returned because: <ul style="list-style-type: none"> ▪ The <i>pAuth</i> object did not pass the firewall rule validations specified in the definition of the FW_AUTH_SET data type. ▪ One of the required values is not specified.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.55 RRPC_FWEnumAuthenticationSets2_10 (Opnum 54)

The **RRPC_FWEnumAuthenticationSets2_10 (Opnum 54)** method requests the server to return all the authentication sets of the specified IPsec phase contained in the store referenced by the *hPolicyStore* handle. The method returns a linked list of these objects. The only method supported is binary version 0x020A.

```

unsigned long RRPC_FWEnumAuthenticationSets2_10(
    [in] FW_CONN_HANDLE rpcConnHandle,
    [in] FW_POLICY_STORE_HANDLE hPolicyStore,
    [in, range(FW_IPSEC_PHASE_INVALID+1, FW_IPSEC_PHASE_MAX-1)]
    FW_IPSEC_PHASE IpSecPhase,
    [in] unsigned long dwFilteredByStatus,
    [in] unsigned short wFlags,
    [out, ref] unsigned long* pdwNumAuthSets,
    [out] PFW_AUTH_SET* ppAuth
);

```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicyStore: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read/write access rights.

IpSecPhase: This parameter specifies the specific IPsec negotiation phase to which this set applies.

dwFilteredByStatus: This parameter is a combination of flags from the [FW_RULE_STATUS_CLASS](#) enumeration. This method uses this bitmask to determine

whether rules should be returned. Rules that contain a status code of the specified class that match this parameter will be returned in the linked list.

wFlags: This parameter is a combination of flags from the [FW_ENUM_RULES_FLAGS](#) enumeration, which modifies the behavior of the method and performs operations on the rules before returning them in the linked list.

pdwNumAuthSets: This is an output parameter that on success MUST be equal to the number of sets returned.

ppAuth: This parameter represents the authentication set that the client adds to the store. The set MUST be valid, as specified in the definition of the [FW_AUTH_SET](#) data type.

Return Values: The method returns 0 if successful; if it fails, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The <i>hPolicyStore</i> handle was not opened with read/write access rights. The error is also returned if the client does not have the required credentials to call the method.
0x00000057 ERROR_INVALID_PARAMETER	The <i>dwProfileFilter</i> parameter contains invalid profiles.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.56 RRPC_FWAddCryptoSet2_10 (Opnum 55)

The **RRPC_FWAddCryptoSet2_10 (Opnum 55)** method adds a cryptographic set in the cryptographic linked list of the memory representation of the store being modified. The only method supported is binary version 0x020A.

```
unsigned long RRPC_FWAddCryptoSet2_10(  
    [in] FW_CONN_HANDLE rpcConnHandle,  
    [in] FW_POLICY_STORE_HANDLE hPolicy,  
    [in] PFW_CRYPTO_SET pCrypto,  
    [out] FW_RULE_STATUS* pStatus  
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read/write access rights.

pCrypto: This parameter represents the cryptographic set that the client adds to the store. The set MUST be valid, as specified in the definition of the [FW_CRYPTO_SET](#) data type.

pStatus: This output parameter is the status code of the rule as specified by the [FW_RULE_STATUS](#) enumeration. This field is filled out on return.

Return Values: The method returns 0 if successful; if it fails, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x000000B7 ERROR_ALREADY_EXISTS	The specified cryptographic set has a cryptographic set ID that already exists in the specified store.
0x00000032 ERROR_NOT_SUPPORTED	The specified store does not support this method; the store might be read-only.
0x00000005 ERROR_ACCESS_DENIED	The <i>hPolicy</i> handle was not opened with read/write access rights. The error is also returned if the client does not have the required credentials to call the method.
0x00000057 ERROR_INVALID_PARAMETER	One of the parameters of this method either is incorrect or is required and not specified. This error can be returned because: <ul style="list-style-type: none">▪ The <i>pCrypto</i> object did not pass the crypto set validations specified in the definition of the FW_CRYPTOSSET data type.▪ One of the required values is not specified.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

This method adds a firewall rule in the firewall linked list of the memory representation of the store being modified. It also writes through and saves the rule on disk. If called on an online store, the firewall rule is also enforced.

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.57 RRPC_FWSetCryptoSet2_10 (Opnum 56)

The **RRPC_FWSetCryptoSet2_10 (Opnum 56)** method requests the server to modify the specified cryptographic set in the policy contained in the policy store referenced by the handle specified in the *hPolicy* parameter. The only method supported is binary version 0x020A.

```
unsigned long RRPC_FWSetCryptoSet2_10(  
    [in] FW_CONN_HANDLE rpcConnHandle,  
    [in] FW_POLICY_STORE_HANDLE hPolicy,  
    [in] PFW_CRYPTOSSET pCrypto,  
    [out] FW_RULE_STATUS* pStatus  
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicy: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the

[RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read/write access rights.

pCrypto: This parameter represents the cryptographic set that the client adds to the store. The set MUST be valid, as specified in the definition of the [FW_CRYPTO_SET](#) data type.

pStatus: This output parameter is the status code of the rule as specified by the [FW_RULE_STATUS](#) enumeration. This field is filled out on return.

Return Values: The method returns 0 if successful; if it fails, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000032 ERROR_NOT_SUPPORTED	The specified store does not support this method; the store might be read-only.
0x00000005 ERROR_ACCESS_DENIED	The <i>hPolicy</i> handle was not opened with read/write access rights. The error is also returned if the client does not have the required credentials to call the method.
0x00000002 ERROR_FILE_NOT_FOUND	The specified rule referenced by the wszSetID member string of the FW_CRYPTO_SET data type is not found in the policy store.
0x00000057 ERROR_INVALID_PARAMETER	One of the parameters of this method either is incorrect or is required and not specified. This error can be returned because: <ul style="list-style-type: none">▪ The <i>pCrypto</i> object did not pass the crypto set validations specified in the definition of the FW_CRYPTO_SET data type.▪ One of the required values is not specified.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.4.58 RRPC_FWEnumCryptoSets2_10 (Opnum 57)

The **RRPC_FWEnumCryptoSets2_10 (Opnum 57)** method requests the server to return all the cryptographic sets of the specified IPsec phase contained in the store referenced by the *hPolicyStore* handle. The method returns a linked list of these objects. The only method supported is binary version 0x020A.

```
unsigned long RRPC_FWEnumCryptoSets2_10(  
    [in] FW_CONN_HANDLE rpcConnHandle,  
    [in] FW_POLICY_STORE_HANDLE hPolicyStore,  
    [in, range(FW_IPSEC_PHASE_INVALID+1, FW_IPSEC_PHASE_MAX-1)]  
    FW_IPSEC_PHASE IpSecPhase,  
    [in] unsigned long dwFilteredByStatus,  
    [in] unsigned short wFlags,  
    [out, ref] unsigned long* pdwNumSets,  
    [out] PFW_CRYPTO_SET* ppCryptoSets  
);
```

rpcConnHandle: This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol.

hPolicyStore: This input parameter is an [FW_POLICY_STORE_HANDLE](#) data type. The data type MUST contain an opened policy store handle, successfully opened with the [RRPC_FWOpenPolicyStore \(Opnum 0\)](#) method. The handle MUST have read/write access rights.

IpSecPhase: This parameter specifies the specific IPsec negotiation phase to which this set applies.

dwFilteredByStatus: This parameter is a combination of flags from the [FW_RULE_STATUS_CLASS](#) enumeration. This method uses this bitmask to determine whether rules should be returned. Rules that contain a status code of the class specified that match this parameter will be returned in the linked list.

wFlags: This parameter is a combination of flags from the [FW_ENUM_RULES_FLAGS](#) enumeration, which modifies the behavior of the method and performs operations on the rules before returning them in the linked list.

pdwNumSets: This is an output parameter that, on success, MUST be equal to the number of sets returned.

ppCryptoSets: This parameter represents the authentication set that the client adds to the store. The set MUST be valid, as specified in the definition of the [FW_AUTH_SET](#) data type.

Return Values: The method returns 0 if successful; if it fails, it returns a nonzero error code. The field can take any specific error code value, as specified in [\[MS-ERREF\]](#). The following return values are common.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The <i>hPolicyStore</i> handle was not opened with read/write access rights. The error is also returned if the client does not have the required credentials to call the method.
0x00000057 ERROR_INVALID_PARAMETER	The <i>dwProfileFilter</i> parameter contains invalid profiles.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [\[MS-RPCE\]](#).

The server MUST validate that the client has appropriate credentials (as defined in section [3.1.4](#)) before executing this method.

3.1.5 Timer Events

No timer events are required on the server other than the events maintained in the underlying RPC transport.

3.1.6 Other Local Events

No additional local events are used on the server other than the events maintained in the underlying RPC transport.

3.2 Client Details

3.2.1 Abstract Data Model

None.

3.2.2 Timers

No protocol timers are required other than those internal ones used in the RPC to implement resiliency to network outages, as specified in [\[MS-RPCE\]](#).

3.2.3 Initialization

The client creates an RPC association (or binding) to the server RPC before an RPC method is called. The client may create a separate association for each method invocation, or it may reuse an association for multiple invocations.

3.2.4 Message Processing Events and Sequencing Rules

This protocol MUST indicate to the RPC runtime that it is to perform a strict NDR data consistency check at target level 6.0, as specified in [\[MS-RPCE\]](#) section 3.

The client SHOULD ignore errors returned from the RPC server and notify the application invoker of the error received. Otherwise, no special message processing is required on the client beyond the processing required in the underlying RPC protocol.

3.2.5 Timer Events

No protocol timer events are required on the client other than those internal ones maintained in the underlying RPC, as specified in [\[MS-RPCE\]](#).

3.2.6 Other Local Events

No local events are required on the client other than those internal ones maintained in the underlying RPC, as specified in [\[MS-RPCE\]](#).

4 Protocol Examples

4.1 Opening a Policy Store

Before a client application can perform most of the operations, it must open a policy store handle. The protocol sequence that opens a policy store is as follows.

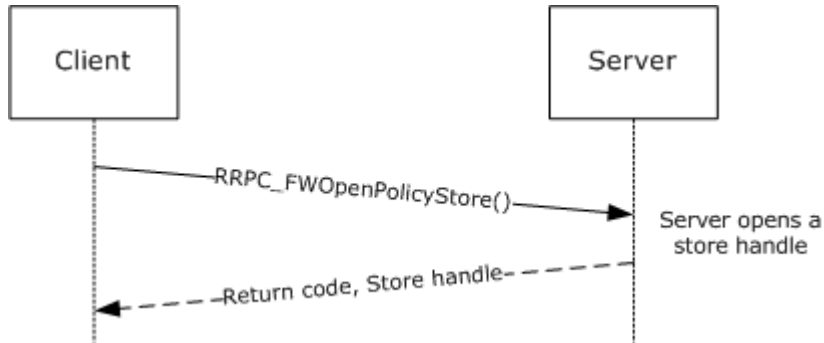


Figure 2: Opening a policy store

To open a policy store, the client must first get an `rpcBinding` to this interface in the server. Then the client simply calls the RPC method to open the required store. In this case, the client chooses the local store.

```
FW_POLICY_STORE_HANDLE    hStore = NULL;
DWORD
RRPC_FWOpenPolicyStore(
    [in] FW_CONN_HANDLE          rpcConnHandle = rpcBinding,
    [in] WORD                    BinaryVersion = 0x0200,
    [in] FW_STORE_TYPE           StoreType = FW_STORE_TYPE_LOCAL,
    [in] FW_POLICY_ACCESS_RIGHT AccessRight = FW_POLICY_ACCESS_RIGHT_READ_WRITE,
    [in] DWORD                   dwFlags = 0,
    [out] PFW_POLICY_STORE_HANDLE phPolicyStore = &hStore
);
```

4.2 Adding a Firewall Rule

Once the client has a handle to an open policy store, the client can perform operations on the policy store. The protocol sequence that adds a firewall rule to the policy store is as follows.

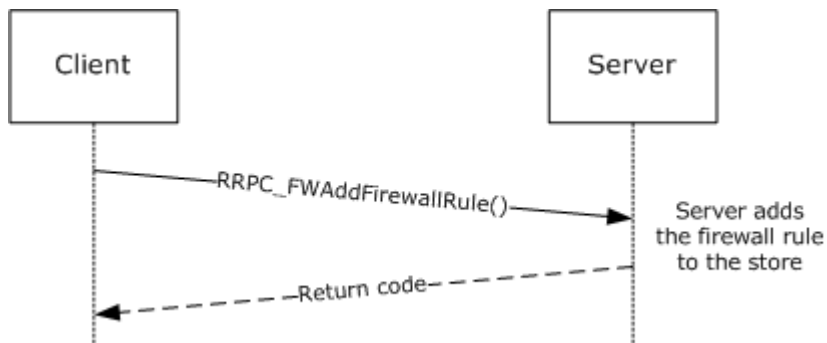


Figure 3: Adding a firewall rule

To add a firewall rule, the client application first fills an [FW_RULE](#) structure. The following examples fill this structure to represent a rule to allow inbound traffic to port 80 for the "c:\servers\MyWebServer.exe" application, which is also a service with the WebServerSVC name. The example also places this rule in the "HTTPWebServer" rule group.

```
FW_PORT_RANGE Port = {80,80};
FW_RULE HTTPRule =
{
    struct _tag_FW_RULE *pNext = NULL;
    WORD          wSchemaVersion = 0x0200;
    WCHAR*        wszRuleId = L"{6b5bdd1e-528c-422c-af8c-a4079be4fe48}";
    WCHAR*        wszName = "Web server requests";
    WCHAR*        wszDescription = "This rule allows incoming HTTP server requests";
    DWORD         dwProfiles = FW_PROFILE_TYPE_ALL;
    FW_DIRECTION  Direction = FW_DIR_IN;
    WORD          wIpProtocol = 0x0006;
    FW_PORTS      LocalPorts = {0x0000, {1, &Port}};
    FW_PORTS      RemotePorts = {0};

    FW_ADDRESSES  LocalAddresses = {0};
    FW_ADDRESSES  RemoteAddresses = {0};
    FW_INTERFACE_LUIDS LocalInterfaceIds = {0};
    DWORD         dwLocalInterfaceTypes = 0;
    WCHAR*        wszLocalApplication = "c:\servers\MyWebServer.exe";
    WCHAR*        wszLocalService = "WebServerSVC";
    FW_RULE_ACTION Action = FW_RULE_ACTION_ALLOW;
    WORD          wFlags = FW_RULE_FLAGS_ACTIVE;

    WCHAR*        wszRemoteMachineAuthorizationList = NULL;
    WCHAR*        wszRemoteUserAuthorizationList = NULL;
    WCHAR*        wszRuleGroupName = L"HTTP WebServer";
    FW_OS_PLATFORM_LIST PlatformValidityList = {0};

    FW_RULE_STATUS Status = FW_RULE_STATUS_OK;
    FW_RULE_ORIGIN_TYPE Origin = 0;
    WCHAR*        wszGPOName = NULL;
    DWORD         Reserved = 0;
};
```

Once the **FW_RULE** structure is filled out, the client can simply invoke the RPC [RRPC FWAddFirewallRule](#) method, passing the required parameters as follows.

```
DWORD
RRPC_FWAddFirewallRule(
    [in] FW_CONN_HANDLE          rpcConnHandle = rpcBinding,
    [in] FW_POLICY_STORE_HANDLE  hPolicyStore = hStore,
    [in] PFW_RULE                pRule = &HTTPRule
);
```

If the return code is **FW_ERROR_ALREADY_EXISTS**, the rule exists in the store. The client may want to try using a different Rule ID or bubble up the error.

4.3 Enumerating the Firewall Rules

To enumerate the firewall rules that the server is enforcing in the store, the client calls the [RRPC_FWEnumFirewallRules \(Opnum 9\)](#) method. The protocol sequence that enumerates firewall rules from the policy store is as follows:

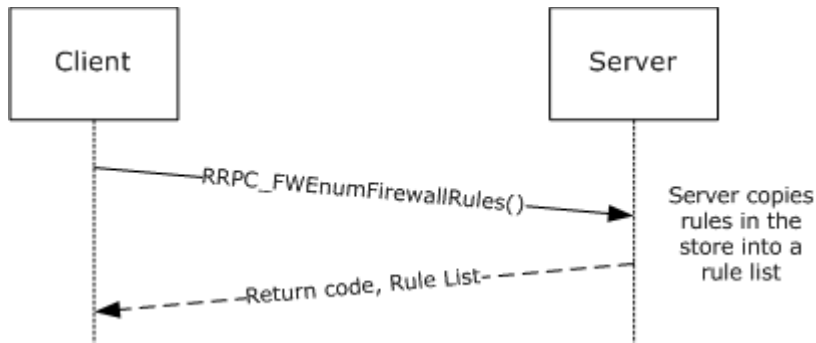


Figure 4: Enumerating firewall rules

In this case example, the client enumerates rules in the current profile and filters by `FW_RULE_STATUS_CLASS_OK` and `FW_RULE_STATUS_CLASS_PARTIALLY_IGNORED`.

```
PFW_RULE pRules = NULL;
DWORD dwNumRules = 0;

DWORD
RRPC_FWEnumFirewallRules(
    [in] FW_CONN_HANDLE      rpcConnHandle = rpcBinding ,
    [in] FW_POLICY_STORE_HANDLE hPolicyStore = hStore,
    [in] DWORD                dwFilteredByStatus =
FW_RULE_STATUS_CLASS_OK | FW_RULE_STATUS_CLASS_PARTIALLY_IGNORED,
    [in] DWORD                dwProfileFilter, FW_PROFILE_TYPE_CURRENT,
    [in] WORD                 wFlag = 0
    [out, ref] DWORD          *pdwNumRules = &dwNumRules,
    [out] PFW_RULE            *ppRules = &pRules
);
```

4.4 Closing a Policy Store Handle

Once a client application has finished managing the policy, it must close the policy store handle. The protocol sequence that closes a policy store follows.

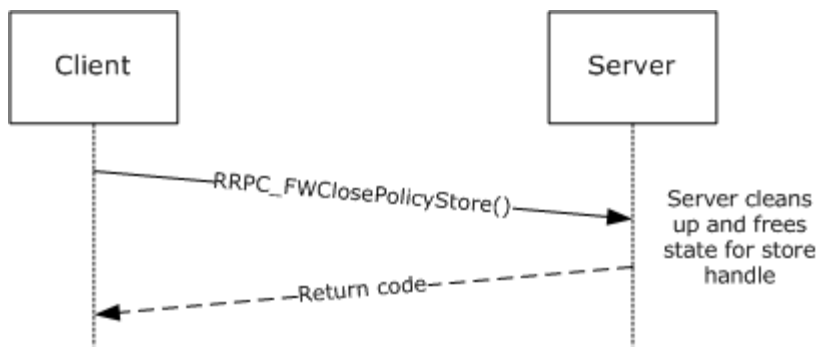


Figure 5: Closing a policy store

To close the handle, the client simply passes the handle to the close method.

```

DWORD
RRPC_FWClosePolicyStore(
    [in] FW_CONN_HANDLE          rpcConnHandle = rpcBinding,
    [in, out] PFW_POLICY_STORE_HANDLE phPolicyStore = &hStore
);
  
```

5 Security

5.1 Security Considerations for Implementers

The enumeration methods require the server to return the correct number of objects linked in the returned linked list. For example, the **DWORD** variable passed in the *pdwNumRules* parameter of [RRPC FWEnumFirewallRules \(Opnum 9\)](#) MUST be equal to the actual number of rules returned in *ppRules*.

However, the client MUST NOT assume that the server is accurate in the actual object count. The client can allocate a buffer based on the rule count; however, while filling the buffer, the client MUST actively validate that the number of objects in the buffer does not exceed the object count. Failure to do this validation could result in buffer overruns on the client.

5.2 Index of Security Parameters

Security Parameter	Section
Remote procedure call (RPC) authentication	2.1
The required permissions to call each of the methods of the protocol interface	3.1.4

6 Appendix A: Full IDL

For ease of implementation, the full IDL is provided below.

```
import "ms-dtyp.idl";

cpp_quote("#ifndef __FIREWALL_H__")
cpp_quote("#define FW_CURRENT_BINARY_VERSION (FW_VERSION(2,0))")
cpp_quote("#define FW_CURRENT_SCHEMA_VERSION (FW_VERSION(2,0))")

typedef enum _tag_FW_STORE_TYPE
{
    FW_STORE_TYPE_INVALID,
    FW_STORE_TYPE_GP_RSOP,
    FW_STORE_TYPE_LOCAL,
    FW_STORE_TYPE_NOT_USED_VALUE_3,
    FW_STORE_TYPE_NOT_USED_VALUE_4,
    FW_STORE_TYPE_DYNAMIC,
    FW_STORE_TYPE_GPO,
    FW_STORE_TYPE_DEFAULTS,
    FW_STORE_TYPE_MAX
} FW_STORE_TYPE;

typedef
[v1_enum]
enum _tag_FW_PROFILE_TYPE
{
    FW_PROFILE_TYPE_INVALID = 0,
    FW_PROFILE_TYPE_DOMAIN = 0x001,
    FW_PROFILE_TYPE_STANDARD = 0x002,
    FW_PROFILE_TYPE_PRIVATE = FW_PROFILE_TYPE_STANDARD,
    FW_PROFILE_TYPE_PUBLIC = 0x004,
    FW_PROFILE_TYPE_ALL = 0x7FFFFFFF,
    FW_PROFILE_TYPE_CURRENT = 0x80000000,
    FW_PROFILE_TYPE_NONE = FW_PROFILE_TYPE_CURRENT + 1
} FW_PROFILE_TYPE;

typedef enum _tag_FW_POLICY_ACCESS_RIGHT
{
    FW_POLICY_ACCESS_RIGHT_INVALID,
    FW_POLICY_ACCESS_RIGHT_READ,
    FW_POLICY_ACCESS_RIGHT_READ_WRITE,
    FW_POLICY_ACCESS_RIGHT_MAX
} FW_POLICY_ACCESS_RIGHT;

typedef struct _tag_FW_IPV4_SUBNET
{
    unsigned long    dwAddress;
    unsigned long    dwSubNetMask;
} FW_IPV4_SUBNET, *PFW_IPV4_SUBNET;

typedef struct _tag_FW_IPV4_SUBNET_LIST
{
    [range(0, 1000)]
    unsigned long    dwNumEntries;
    [size_is(dwNumEntries)]
    PFW_IPV4_SUBNET pSubNets;
}
```

```

} FW_IPV4_SUBNET_LIST, *PFW_IPV4_SUBNET_LIST;

typedef struct _tag_FW_IPV6_SUBNET
{
    unsigned char            Address[16];
    [range(0, 128)]
    unsigned long            dwNumPrefixBits;
} FW_IPV6_SUBNET, *PFW_IPV6_SUBNET;

typedef struct _tag_FW_IPV6_SUBNET_LIST
{
    [range(0, 1000)]
    unsigned long            dwNumEntries;
    [size_is(dwNumEntries)]
    PFW_IPV6_SUBNET          pSubNets;
} FW_IPV6_SUBNET_LIST, *PFW_IPV6_SUBNET_LIST;

typedef struct _tag_FW_IPV4_ADDRESS_RANGE
{
    unsigned long            dwBegin;
    unsigned long            dwEnd;
} FW_IPV4_ADDRESS_RANGE, *PFW_IPV4_ADDRESS_RANGE;

typedef struct _tag_FW_IPV6_ADDRESS_RANGE
{
    unsigned char            Begin[16];
    unsigned char            End[16];
} FW_IPV6_ADDRESS_RANGE, *PFW_IPV6_ADDRESS_RANGE;

typedef struct _tag_FW_IPV4_RANGE_LIST
{
    [range(0, 1000)]
    unsigned long            dwNumEntries;
    [size_is(dwNumEntries)]
    PFW_IPV4_ADDRESS_RANGE    pRanges;
} FW_IPV4_RANGE_LIST, *PFW_IPV4_RANGE_LIST;

typedef struct _tag_FW_IPV6_RANGE_LIST
{
    [range(0, 1000)]
    unsigned long            dwNumEntries;
    [size_is(dwNumEntries)]
    PFW_IPV6_ADDRESS_RANGE    pRanges;
} FW_IPV6_RANGE_LIST, *PFW_IPV6_RANGE_LIST;

typedef struct _tag_FW_PORT_RANGE
{
    unsigned short           wBegin;
    unsigned short           wEnd;
} FW_PORT_RANGE, *PFW_PORT_RANGE;

typedef struct _tag_FW_PORT_RANGE_LIST
{
    [range(0, 1000)]
    unsigned long            dwNumEntries;
    [size_is(dwNumEntries)]
    PFW_PORT_RANGE          pPorts;
} FW_PORT_RANGE_LIST, *PFW_PORT_RANGE_LIST;

```

```

typedef enum _tag_FW_PORT_KEYWORD
{
    FW_PORT_KEYWORD_NONE           = 0x00,
    FW_PORT_KEYWORD_DYNAMIC_RPC_PORTS = 0x01,
    FW_PORT_KEYWORD_RPC_EP         = 0x02,
    FW_PORT_KEYWORD_TEREDO_PORT    = 0x04,
    FW_PORT_KEYWORD_IP_TLS_IN      = 0x08,
    FW_PORT_KEYWORD_IP_TLS_OUT     = 0x10,
    FW_PORT_KEYWORD_MAX            = 0x20,
    FW_PORT_KEYWORD_MAX_V2_1       = 0x08
} FW_PORT_KEYWORD;

typedef struct _tag_FW_PORTS
{
    unsigned short          wPortKeywords;
    FW_PORT_RANGE_LIST Ports;
} FW_PORTS, *PFW_PORTS;

cpp_quote("#define FW_ICMP_CODE_ANY (256)")
cpp_quote("#define FW_IP_PROTOCOL_ANY (256)")

typedef struct _tag_FW_ICMP_TYPE_CODE
{
    unsigned char          bType;
    [range(0, 256)]
    unsigned short        wCode;
} FW_ICMP_TYPE_CODE, *PFW_ICMP_TYPE_CODE;

typedef struct _tag_FW_ICMP_TYPE_CODE_LIST
{
    [range(0, 1000)]
    unsigned long          dwNumEntries;
    [size_is(dwNumEntries)]
    PFW_ICMP_TYPE_CODE pEntries;
} FW_ICMP_TYPE_CODE_LIST, *PFW_ICMP_TYPE_CODE_LIST;

typedef struct _tag_FW_INTERFACE_LUIDS
{
    [range(0, 1000)]
    unsigned long          dwNumLUIDs;
    [size_is(dwNumLUIDs)]
    GUID*                  pLUIDs;
} FW_INTERFACE_LUIDS, *PFW_INTERFACE_LUIDS;

typedef enum _tag_FW_DIRECTION
{
    FW_DIR_INVALID = 0,
    FW_DIR_IN,
    FW_DIR_OUT,
    FW_DIR_MAX
} FW_DIRECTION;

typedef enum _tag_FW_INTERFACE_TYPE
{
    FW_INTERFACE_TYPE_ALL           = 0x0000,
    FW_INTERFACE_TYPE_LAN          = 0x0001,
    FW_INTERFACE_TYPE_WIRELESS     = 0x0002,

```

```

    FW_INTERFACE_TYPE_REMOTE_ACCESS = 0x0004,
    FW_INTERFACE_TYPE_MAX           = 0x0008
} FW_INTERFACE_TYPE;

typedef enum _tag_FW_ADDRESS_KEYWORD
{
    FW_ADDRESS_KEYWORD_NONE           = 0x0000,
    FW_ADDRESS_KEYWORD_LOCAL_SUBNET   = 0x0001,
    FW_ADDRESS_KEYWORD_DNS            = 0x0002,
    FW_ADDRESS_KEYWORD_DHCP           = 0x0004,
    FW_ADDRESS_KEYWORD_WINS           = 0x0008,
    FW_ADDRESS_KEYWORD_DEFAULT_GATEWAY = 0x0010,
    FW_ADDRESS_KEYWORD_MAX            = 0x0020
}FW_ADDRESS_KEYWORD;

typedef struct _tag_FW_ADDRESSES
{
    unsigned long          dwV4AddressKeywords;
    unsigned long          dwV6AddressKeywords;
    FW_IPV4_SUBNET_LIST   V4SubNets;
    FW_IPV4_RANGE_LIST    V4Ranges;
    FW_IPV6_SUBNET_LIST   V6SubNets;
    FW_IPV6_RANGE_LIST    V6Ranges;
}FW_ADDRESSES, *PFW_ADDRESSES;

typedef
[v1_enum]
enum _tag_FW_RULE_STATUS
{
    FW_RULE_STATUS_OK = 0x00010000,
    FW_RULE_STATUS_PARTIALLY_IGNORED = 0x00020000,
    FW_RULE_STATUS_IGNORED = 0x00040000,
    FW_RULE_STATUS_PARSING_ERROR_NAME = 0x00080001,
    FW_RULE_STATUS_PARSING_ERROR_DESC = 0x00080002,
    FW_RULE_STATUS_PARSING_ERROR_APP = 0x00080003,
    FW_RULE_STATUS_PARSING_ERROR_SVC = 0x00080004,
    FW_RULE_STATUS_PARSING_ERROR_RMA = 0x00080005,
    FW_RULE_STATUS_PARSING_ERROR_RUA = 0x00080006,
    FW_RULE_STATUS_PARSING_ERROR_EMBD = 0x00080007,
    FW_RULE_STATUS_PARSING_ERROR_RULE_ID = 0x00080008,
    FW_RULE_STATUS_PARSING_ERROR_PHASE1_AUTH = 0x00080009,
    FW_RULE_STATUS_PARSING_ERROR_PHASE2_CRYPT0 = 0x0008000A,
    FW_RULE_STATUS_PARSING_ERROR_PHASE2_AUTH = 0x0008000B,
    FW_RULE_STATUS_PARSING_ERROR_RESOLVE_APP = 0x0008000C,
    FW_RULE_STATUS_PARSING_ERROR_MAINMODE_ID = 0x0008000D,
    FW_RULE_STATUS_PARSING_ERROR_PHASE1_CRYPT0 = 0x0008000E,
    FW_RULE_STATUS_PARSING_ERROR = 0x00080000,
    FW_RULE_STATUS_SEMANTIC_ERROR_RULE_ID = 0x00100010,
    FW_RULE_STATUS_SEMANTIC_ERROR_PORTS = 0x00100020,
    FW_RULE_STATUS_SEMANTIC_ERROR_PORT_KEYW = 0x00100021,
    FW_RULE_STATUS_SEMANTIC_ERROR_PORT_RANGE = 0x00100022,
    FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_V4_SUBNETS = 0x00100040,
    FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_V6_SUBNETS = 0x00100041,
    FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_V4_RANGES = 0x00100042,
    FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_V6_RANGES = 0x00100043,
    FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_RANGE = 0x00100044,
    FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_MASK = 0x00100045,
    FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_PREFIX = 0x00100046,
    FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_KEYW = 0x00100047,

```

```

FW_RULE_STATUS_SEMANTIC_ERROR_LADDR_PROP           = 0x00100048,
FW_RULE_STATUS_SEMANTIC_ERROR_RADDR_PROP           = 0x00100049,
FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_V6             = 0x0010004A,
FW_RULE_STATUS_SEMANTIC_ERROR_LADDR_INTF          = 0x0010004B,
FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_V4             = 0x0010004C,
FW_RULE_STATUS_SEMANTIC_ERROR_TUNNEL_ENDPOINT_ADDR = 0x0010004D,
FW_RULE_STATUS_SEMANTIC_ERROR_DTE_VER             = 0x0010004E,
FW_RULE_STATUS_SEMANTIC_ERROR_DTE_MISMATCH_ADDR   = 0x0010004F,
FW_RULE_STATUS_SEMANTIC_ERROR_PROFILE             = 0x00100050,
FW_RULE_STATUS_SEMANTIC_ERROR_ICMP               = 0x00100060,
FW_RULE_STATUS_SEMANTIC_ERROR_ICMP_CODE          = 0x00100061,
FW_RULE_STATUS_SEMANTIC_ERROR_IF_ID              = 0x00100070,
FW_RULE_STATUS_SEMANTIC_ERROR_IF_TYPE            = 0x00100071,
FW_RULE_STATUS_SEMANTIC_ERROR_ACTION              = 0x00100080,
FW_RULE_STATUS_SEMANTIC_ERROR_ALLOW_BYPASS        = 0x00100081,
FW_RULE_STATUS_SEMANTIC_ERROR_DO_NOT_SECURE       = 0x00100082,
FW_RULE_STATUS_SEMANTIC_ERROR_ACTION_BLOCK_IS_ENCRYPTED_SECURE
                                                    = 0x00100083,
FW_RULE_STATUS_SEMANTIC_ERROR_DIR                 = 0x00100090,
FW_RULE_STATUS_SEMANTIC_ERROR_PROT                = 0x001000A0,
FW_RULE_STATUS_SEMANTIC_ERROR_PROT_PROP           = 0x001000A1,
FW_RULE_STATUS_SEMANTIC_ERROR_DEFER_EDGE_PROP     = 0x001000A2,
FW_RULE_STATUS_SEMANTIC_ERROR_ALLOW_BYPASS_OUTBOUND = 0x001000A3,
FW_RULE_STATUS_SEMANTIC_ERROR_DEFER_USER_INVALID_RULE
                                                    = 0x001000A4,
FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS               = 0x001000B0,
FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_AUTO_AUTH     = 0x001000B1,
FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_AUTO_BLOCK    = 0x001000B2,
FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_AUTO_DYN_RPC  = 0x001000B3,
FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_AUTHENTICATE_ENCRYPT
                                                    = 0x001000B4,
FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_AUTH_WITH_ENC_NEGOTIATE_VER
                                                    = 0x001000B5,
FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_AUTH_WITH_ENC_NEGOTIATE
                                                    = 0x001000B6,
FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_ESP_NO_ENCAP_VER
                                                    = 0x001000B7,
FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_ESP_NO_ENCAP
                                                    = 0x001000B8,
FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_TUNNEL_AUTH_MODES_VER
                                                    = 0x001000B9,
FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_TUNNEL_AUTH_MODES
                                                    = 0x001000BA,
FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_IP_TLS_VER    = 0x001000BB,
FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_PORTRANGE_VER
                                                    = 0x001000BC,
FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_ADDRS_TRAVERSE_DEFER_VER
                                                    = 0x001000BD,
FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_AUTH_WITH_ENC_NEGOTIATE_OUTBOUND
                                                    = 0x001000BE,
FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_AUTHENTICATE_WITH_OUTBOUND_BYPASS_VER
                                                    = 0x001000BF,
FW_RULE_STATUS_SEMANTIC_ERROR_REMOTE_AUTH_LIST    = 0x001000C0,
FW_RULE_STATUS_SEMANTIC_ERROR_REMOTE_USER_LIST    = 0x001000C1,
FW_RULE_STATUS_SEMANTIC_ERROR_PLATFORM            = 0x001000E0,
FW_RULE_STATUS_SEMANTIC_ERROR_PLATFORM_OP_VER     = 0x001000E1,
FW_RULE_STATUS_SEMANTIC_ERROR_PLATFORM_OP        = 0x001000E2,

```

```

FW_RULE_STATUS_SEMANTIC_ERROR_DTE_NOANY_ADDR           = 0x001000F0,
FW_RULE_STATUS_SEMANTIC_TUNNEL_EXEMPT_WITH_GATEWAY     = 0x001000F1,
FW_RULE_STATUS_SEMANTIC_TUNNEL_EXEMPT_VER             = 0x001000F2,

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_AUTH_SET_ID      = 0x00100500,
FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPT0_SET_ID    = 0x00100510,
FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPT0_SET_ID    = 0x00100511,
FW_RULE_STATUS_SEMANTIC_ERROR_ERROR_SET_ID           = 0x00101000,
FW_RULE_STATUS_SEMANTIC_ERROR_IPSEC_PHASE             = 0x00101010,
FW_RULE_STATUS_SEMANTIC_ERROR_EMPTY_SUITES           = 0x00101020,
FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_AUTH_METHOD     = 0x00101030,
FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_AUTH_METHOD     = 0x00101031,
FW_RULE_STATUS_SEMANTIC_ERROR_AUTH_METHOD_ANONYMOUS  = 0x00101032,
FW_RULE_STATUS_SEMANTIC_ERROR_AUTH_METHOD_DUPLICATE  = 0x00101033,
FW_RULE_STATUS_SEMANTIC_ERROR_AUTH_SUITE_FLAGS       = 0x00101040,
FW_RULE_STATUS_SEMANTIC_ERROR_HEALTH_CERT            = 0x00101041,
FW_RULE_STATUS_SEMANTIC_ERROR_AUTH_SIGNCERT_VER      = 0x00101042,
FW_RULE_STATUS_SEMANTIC_ERROR_AUTH_INTERMEDIATE_CA_VER = 0x00101043,

FW_RULE_STATUS_SEMANTIC_ERROR_MACHINE_SHKEY          = 0x00101050,
FW_RULE_STATUS_SEMANTIC_ERROR_CA_NAME                = 0x00101060,
FW_RULE_STATUS_SEMANTIC_ERROR_MIXED_CERTS            = 0x00101061,
FW_RULE_STATUS_SEMANTIC_ERROR_NON_CONTIGUOUS_CERTS   = 0x00101062,
FW_RULE_STATUS_SEMANTIC_ERROR_MIXED_CA_TYPE_IN_BLOCK = 0x00101063,
FW_RULE_STATUS_SEMANTIC_ERROR_MACHINE_USER_AUTH     = 0x00101070,
FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPT0_NON_DEFAULT_ID = 0x00105000,

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPT0_FLAGS    = 0x00105001,
FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPT0_TIMEOUT_MINUTES = 0x00105002,

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPT0_TIMEOUT_SESSIONS = 0x00105003,

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPT0_KEY_EXCHANGE = 0x00105004,

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPT0_ENCRYPTION = 0x00105005,

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPT0_HASH     = 0x00105006,

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPT0_ENCRYPTION_VER = 0x00105007,

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPT0_HASH_VER = 0x00105008,

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPT0_PFS      = 0x00105020,

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPT0_PROTOCOL = 0x00105021,

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPT0_ENCRYPTION = 0x00105022,

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPT0_HASH     = 0x00105023,

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPT0_TIMEOUT_MINUTES = 0x00105024,

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPT0_TIMEOUT_KBYTES = 0x00105025,

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPT0_ENCRYPTION_VER = 0x00105026,

```



```

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPT_HASH_VER
FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_OR_AND_CONDITIONS
FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_AND_CONDITIONS
FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_CONDITION_KEY
FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_CONDITION_MATCH_TYPE
FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_CONDITION_DATA_TYPE
FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_CONDITION_KEY_AND_DATA_TYPE
FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_KEYS_PROTOCOL_PORT
FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_KEY_PROFILE
FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_KEY_STATUS
FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_KEY_FILTERID
FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_KEY_APP_PATH
FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_KEY_PROTOCOL
FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_KEY_LOCAL_PORT
FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_KEY_REMOTE_PORT
FW_RULE_STATUS_SEMANTIC_ERROR_QUERY_KEY_SVC_NAME
FW_RULE_STATUS_SEMANTIC_ERROR_REQUIRE_IN_CLEAR_OUT_ON_TRANSPORT
FW_RULE_STATUS_SEMANTIC_ERROR_TUNNEL_BYPASS_TUNNEL_IF_SECURE_ON_TRANSPORT
FW_RULE_STATUS_SEMANTIC_ERROR_AUTH_NOENCAP_ON_TUNNEL
FW_RULE_STATUS_SEMANTIC_ERROR_AUTH_NOENCAP_ON_PSK
FW_RULE_STATUS_SEMANTIC_ERROR_CRYPTO_ENCR_HASH
FW_RULE_STATUS_SEMANTIC_ERROR_CRYPTO_ENCR_HASH_COMPAT
FW_RULE_STATUS_SEMANTIC_ERROR_SCHEMA_VERSION
FW_RULE_STATUS_SEMANTIC_ERROR
FW_RULE_STATUS_RUNTIME_ERROR_PHASE1_AUTH_NOT_FOUND
FW_RULE_STATUS_RUNTIME_ERROR_PHASE2_AUTH_NOT_FOUND
FW_RULE_STATUS_RUNTIME_ERROR_PHASE2_CRYPTO_NOT_FOUND
FW_RULE_STATUS_RUNTIME_ERROR_AUTH_MCHN_SHKEY_MISMATCH
FW_RULE_STATUS_RUNTIME_ERROR_PHASE1_CRYPTO_NOT_FOUND

```

```

FW_RULE_STATUS_RUNTIME_ERROR_AUTH_NOENCAP_ON_TUNNEL = 0x00200006,
FW_RULE_STATUS_RUNTIME_ERROR_AUTH_NOENCAP_ON_PSK    = 0x00200007,

FW_RULE_STATUS_RUNTIME_ERROR                        = 0x00200000,
FW_RULE_STATUS_ERROR = FW_RULE_STATUS_PARSING_ERROR |
    FW_RULE_STATUS_SEMANTIC_ERROR | FW_RULE_STATUS_RUNTIME_ERROR,

FW_RULE_STATUS_ALL                                = 0xFFFF0000
} FW_RULE_STATUS;

typedef enum _tag_FW_RULE_STATUS_CLASS
{
FW_RULE_STATUS_CLASS_OK                          = FW_RULE_STATUS_OK,
FW_RULE_STATUS_CLASS_PARTIALLY_IGNORED          = FW_RULE_STATUS_PARTIALLY_IGNORED,
FW_RULE_STATUS_CLASS_IGNORED                    = FW_RULE_STATUS_IGNORED,
FW_RULE_STATUS_CLASS_PARSING_ERROR              = FW_RULE_STATUS_PARSING_ERROR,
FW_RULE_STATUS_CLASS_SEMANTIC_ERROR            = FW_RULE_STATUS_SEMANTIC_ERROR,
FW_RULE_STATUS_CLASS_RUNTIME_ERROR              = FW_RULE_STATUS_RUNTIME_ERROR,
FW_RULE_STATUS_CLASS_ERROR                      = FW_RULE_STATUS_ERROR,
FW_RULE_STATUS_CLASS_ALL                        = FW_RULE_STATUS_ALL
} FW_RULE_STATUS_CLASS;

typedef enum _tag_FW_OBJECT_CTRL_FLAG
{
FW_OBJECT_CTRL_FLAG_INCLUDE_METADATA           = 0x0001,
} FW_OBJECT_CTRL_FLAG;

typedef enum _tag_FW_ENFORCEMENT_STATE
{
FW_ENFORCEMENT_STATE_INVALID,
FW_ENFORCEMENT_STATE_FULL,
FW_ENFORCEMENT_STATE_WF_OFF_IN_PROFILE,
FW_ENFORCEMENT_STATE_CATEGORY_OFF,
FW_ENFORCEMENT_STATE_DISABLED_OBJECT,
FW_ENFORCEMENT_STATE_INACTIVE_PROFILE,
FW_ENFORCEMENT_STATE_LOCAL_ADDRESS_RESOLUTION_EMPTY,
FW_ENFORCEMENT_STATE_REMOTE_ADDRESS_RESOLUTION_EMPTY,
FW_ENFORCEMENT_STATE_LOCAL_PORT_RESOLUTION_EMPTY,
FW_ENFORCEMENT_STATE_REMOTE_PORT_RESOLUTION_EMPTY,
FW_ENFORCEMENT_STATE_INTERFACE_RESOLUTION_EMPTY,
FW_ENFORCEMENT_STATE_APPLICATION_RESOLUTION_EMPTY,
FW_ENFORCEMENT_STATE_REMOTE_MACHINE_EMPTY,
FW_ENFORCEMENT_STATE_REMOTE_USER_EMPTY,
FW_ENFORCEMENT_STATE_LOCAL_GLOBAL_OPEN_PORTS_DISALLOWED,
FW_ENFORCEMENT_STATE_LOCAL_AUTHORIZED_APPLICATIONS_DISALLOWED,
FW_ENFORCEMENT_STATE_LOCAL_FIREWALL_RULES_DISALLOWED,
FW_ENFORCEMENT_STATE_LOCAL_CONSEC_RULES_DISALLOWED,
FW_ENFORCEMENT_STATE_MISMATCHED_PLATFORM,
FW_ENFORCEMENT_STATE_OPTIMIZED_OUT,
FW_ENFORCEMENT_STATE_MAX
} FW_ENFORCEMENT_STATE;

typedef struct _tag_FW_OBJECT_METADATA
{

```

```

    unsigned __int64 qwFilterContextID;
    [range(0, 100)]
    unsigned long dwNumEntries;
    [size_is(dwNumEntries)]
    FW_ENFORCEMENT_STATE *pEnforcementStates;
} FW_OBJECT_METADATA, *PFW_OBJECT_METADATA;

typedef enum _tag_FW_OS_PLATFORM_OP
{
    FW_OS_PLATFORM_OP_EQ,
    FW_OS_PLATFORM_OP_GTEQ,
    FW_OS_PLATFORM_OP_MAX,
} FW_OS_PLATFORM_OP;

typedef struct _tag_FW_OS_PLATFORM
{
    unsigned char    bPlatform;
    unsigned char    bMajorVersion;
    unsigned char    bMinorVersion;
    unsigned char    Reserved;
} FW_OS_PLATFORM, *PFW_OS_PLATFORM;

typedef struct _tag_FW_OS_PLATFORM_LIST
{
    [range(0, 1000)]
    unsigned long    dwNumEntries;
    [size_is(dwNumEntries)]
    PFW_OS_PLATFORM    pPlatforms;
} FW_OS_PLATFORM_LIST, *PFW_OS_PLATFORM_LIST;

typedef enum _tag_FW_RULE_ORIGIN_TYPE
{
    FW_RULE_ORIGIN_INVALID,
    FW_RULE_ORIGIN_LOCAL,
    FW_RULE_ORIGIN_GP,
    FW_RULE_ORIGIN_DYNAMIC,
    FW_RULE_ORIGIN_AUTOGEN,
    FW_RULE_ORIGIN_HARDCODED,
    FW_RULE_ORIGIN_MAX
} FW_RULE_ORIGIN_TYPE;

typedef enum _tag_FW_ENUM_RULES_FLAGS
{
    FW_ENUM_RULES_FLAG_NONE                = 0x0000,
    FW_ENUM_RULES_FLAG_RESOLVE_NAME        = 0x0001,
    FW_ENUM_RULES_FLAG_RESOLVE_DESCRIPTION = 0x0002,
    FW_ENUM_RULES_FLAG_RESOLVE_APPLICATION = 0x0004,
    FW_ENUM_RULES_FLAG_RESOLVE_KEYWORD     = 0x0008,
    FW_ENUM_RULES_FLAG_RESOLVE_GPO_NAME    = 0x0010,
    FW_ENUM_RULES_FLAG_EFFECTIVE           = 0x0020,
    FW_ENUM_RULES_FLAG_INCLUDE_METADATA    = 0x0040,
    FW_ENUM_RULES_FLAG_MAX                 = 0x0080
} FW_ENUM_RULES_FLAGS;

typedef enum _tag_FW_RULE_ACTION
{
    FW_RULE_ACTION_INVALID = 0,
    FW_RULE_ACTION_ALLOW_BYPASS,

```

```

    FW_RULE_ACTION_BLOCK,
    FW_RULE_ACTION_ALLOW,
    FW_RULE_ACTION_MAX
} FW_RULE_ACTION;

typedef enum _tag_FW_RULE_FLAGS
{
    FW_RULE_FLAGS_NONE = 0x0000,
    FW_RULE_FLAGS_ACTIVE = 0x0001,
    FW_RULE_FLAGS_AUTHENTICATE = 0x0002,
    FW_RULE_FLAGS_AUTHENTICATE_WITH_ENCRYPTION = 0x0004,
    FW_RULE_FLAGS_ROUTEABLE_ADDRS_TRAVERSE = 0x0008,
    FW_RULE_FLAGS_LOOSE_SOURCE_MAPPED = 0x0010,
    FW_RULE_FLAGS_AUTH_WITH_NO_ENCAPSULATION = 0x0020,
    FW_RULE_FLAGS_AUTH_WITH_ENC_NEGOTIATE = 0x0040,
    FW_RULE_FLAGS_ROUTEABLE_ADDRS_TRAVERSE_DEFER_APP = 0x0080,
    FW_RULE_FLAGS_ROUTEABLE_ADDRS_TRAVERSE_DEFER_USER = 0x0100,
    FW_RULE_FLAGS_AUTHENTICATE_BYPASS_OUTBOUND = 0x0200,
    FW_RULE_FLAGS_MAX = 0x0400,
    FW_RULE_FLAGS_MAX_V2_1 = 0x0020,
    FW_RULE_FLAGS_MAX_V2_9 = 0x0040,
} FW_RULE_FLAGS;

typedef struct _tag_FW_RULE2_0
{
    struct _tag_FW_RULE2_0 *pNext;
    unsigned short wSchemaVersion;
    [string, range(1,10001), ref]
    wchar_t* wszRuleId;
    [string, range(1,10001)]
    wchar_t* wszName;
    [string, range(1,10001)]
    wchar_t* wszDescription;
    unsigned long dwProfiles;
    [range(FW_DIR_INVALID, FW_DIR_OUT)]
    FW_DIRECTION Direction;
    [range(0,256)]
    unsigned short wIpProtocol;
    [switch_type(unsigned short), switch_is(wIpProtocol)]
    union
    {
        [case(6,17)]
        struct
        {
            FW_PORTS LocalPorts;
            FW_PORTS RemotePorts;
        };

        [case(1)]
        FW_ICMP_TYPE_CODE_LIST V4TypeCodeList;
        [case(58)]
        FW_ICMP_TYPE_CODE_LIST V6TypeCodeList;
        [default]
        ;
    };

    FW_ADDRESSES LocalAddresses;
    FW_ADDRESSES RemoteAddresses;
}

```

```

FW_INTERFACE_LUIDS LocalInterfaceIds;
unsigned long      dwLocalInterfaceTypes;
[string, range(1,10001)]
wchar_t*          wszLocalApplication;
[string, range(1,10001)]
wchar_t*          wszLocalService;
[range(FW_RULE_ACTION_INVALID, FW_RULE_ACTION_MAX)]
FW_RULE_ACTION    Action;
unsigned short    wFlags;
[string, range(1,10001)]
wchar_t*          wszRemoteMachineAuthorizationList;
[string, range(1,10001)]
wchar_t*          wszRemoteUserAuthorizationList;
[string, range(1,10001)]
wchar_t*          wszEmbeddedContext;
FW_OS_PLATFORM_LIST PlatformValidityList;
FW_RULE_STATUS    Status;
[range(FW_RULE_ORIGIN_INVALID, FW_RULE_ORIGIN_MAX)]
FW_RULE_ORIGIN_TYPE Origin;
[string, range(1,10001)]
wchar_t*          wszGPOName;
unsigned long     Reserved;
} FW_RULE2_0, *PFW_RULE2_0;

```

```

typedef struct _tag_FW_RULE
{
    struct _tag_FW_RULE *pNext;
    unsigned short      wSchemaVersion;
    [string, range(1,512), ref]
    wchar_t*            wszRuleId;
    [string, range(1,10001)]
    wchar_t*            wszName;
    [string, range(1,10001)]
    wchar_t*            wszDescription;
    unsigned long       dwProfiles;
    [range(FW_DIR_INVALID, FW_DIR_OUT)]
    FW_DIRECTION        Direction;
    [range(0,256)]
    unsigned short      wIpProtocol;
    [switch_type(unsigned short), switch_is(wIpProtocol)]
    union
    {
        [case(6,17)]
        struct
        {
            FW_PORTS      LocalPorts;
            FW_PORTS      RemotePorts;
        };

        [case(1)]
        FW_ICMP_TYPE_CODE_LIST V4TypeCodeList;
        [case(58)]
        FW_ICMP_TYPE_CODE_LIST V6TypeCodeList;
        [default]
        ;
    };
};

```

```

FW_ADDRESSES         LocalAddresses;
FW_ADDRESSES         RemoteAddresses;
FW_INTERFACE_LUIDS   LocalInterfaceIds;
unsigned long        dwLocalInterfaceTypes;
[string, range(1,10001)]
wchar_t*             wszLocalApplication;
[string, range(1,10001)]
wchar_t*             wszLocalService;
[range(FW_RULE_ACTION_INVALID, FW_RULE_ACTION_MAX)]
FW_RULE_ACTION       Action;
unsigned short       wFlags;
[string, range(1,10001)]
wchar_t*             wszRemoteMachineAuthorizationList;
[string, range(1,10001)]
wchar_t*             wszRemoteUserAuthorizationList;
[string, range(1,10001)]
wchar_t*             wszEmbeddedContext;
FW_OS_PLATFORM_LIST PlatformValidityList;
FW_RULE_STATUS       Status;
[range(FW_RULE_ORIGIN_INVALID, FW_RULE_ORIGIN_MAX)]
FW_RULE_ORIGIN_TYPE Origin;
[string, range(1,10001)]
wchar_t*             wszGPOName;
unsigned long        Reserved;
[size_is((Reserved & FW_OBJECT_CTRL_FLAG_INCLUDE_METADATA) ? 1 : 0)]
PFW_OBJECT_METADATA pMetaData;
} FW_RULE, *PFW_RULE;

#define FW_PROFILE_CONFIG_LOG_FILE_SIZE_MIN      1
#define FW_PROFILE_CONFIG_LOG_FILE_SIZE_MAX     32767

typedef enum _tag_FW_PROFILE_CONFIG
{
    FW_PROFILE_CONFIG_INVALID,
    FW_PROFILE_CONFIG_ENABLE_FW,
    FW_PROFILE_CONFIG_DISABLE_STEALTH_MODE,
    FW_PROFILE_CONFIG_SHIELDED,
    FW_PROFILE_CONFIG_DISABLE_UNICAST_RESPONSES_TO_MULTICAST_BROADCAST,
    FW_PROFILE_CONFIG_LOG_DROPPED_PACKETS,
    FW_PROFILE_CONFIG_LOG_SUCCESS_CONNECTIONS,
    FW_PROFILE_CONFIG_LOG_IGNORED_RULES,
    FW_PROFILE_CONFIG_LOG_MAX_FILE_SIZE,
    FW_PROFILE_CONFIG_LOG_FILE_PATH,
    FW_PROFILE_CONFIG_DISABLE_INBOUND_NOTIFICATIONS,
    FW_PROFILE_CONFIG_AUTH_APPS_ALLOW_USER_PREF_MERGE,
    FW_PROFILE_CONFIG_GLOBAL_PORTS_ALLOW_USER_PREF_MERGE,
    FW_PROFILE_CONFIG_ALLOW_LOCAL_POLICY_MERGE,
    FW_PROFILE_CONFIG_ALLOW_LOCAL_IPSEC_POLICY_MERGE,
    FW_PROFILE_CONFIG_DISABLED_INTERFACES,
    FW_PROFILE_CONFIG_DEFAULT_OUTBOUND_ACTION,
    FW_PROFILE_CONFIG_DEFAULT_INBOUND_ACTION,
    FW_PROFILE_CONFIG_MAX
} FW_PROFILE_CONFIG;

typedef enum _FW_GLOBAL_CONFIG_IPSEC_EXEMPT_VALUES
{
    FW_GLOBAL_CONFIG_IPSEC_EXEMPT_NONE          = 0x0000,
    FW_GLOBAL_CONFIG_IPSEC_EXEMPT_NEIGHBOR_DISC = 0x0001,
    FW_GLOBAL_CONFIG_IPSEC_EXEMPT_ICMP         = 0x0002,

```

```

    FW_GLOBAL_CONFIG_IPSEC_EXEMPT_ROUTER_DISC    = 0x0004,
    FW_GLOBAL_CONFIG_IPSEC_EXEMPT_DHCP          = 0x0008,
    FW_GLOBAL_CONFIG_IPSEC_EXEMPT_MAX           = 0x0010,
    FW_GLOBAL_CONFIG_IPSEC_EXEMPT_MAX_V2_0      = 0x0004
} FW_GLOBAL_CONFIG_IPSEC_EXEMPT_VALUES;

typedef enum _FW_GLOBAL_CONFIG_PRESHARED_KEY_ENCODING_VALUES
{
    FW_GLOBAL_CONFIG_PRESHARED_KEY_ENCODING_NONE    = 0,
    FW_GLOBAL_CONFIG_PRESHARED_KEY_ENCODING_UTF_8,
    FW_GLOBAL_CONFIG_PRESHARED_KEY_ENCODING_MAX
} FW_GLOBAL_CONFIG_PRESHARED_KEY_ENCODING_VALUES;

typedef enum _FW_GLOBAL_CONFIG_IPSEC_THROUGH_NAT_VALUES
{
    FW_GLOBAL_CONFIG_IPSEC_THROUGH_NAT_NEVER                = 0,
    FW_GLOBAL_CONFIG_IPSEC_THROUGH_NAT_SERVER_BEHIND_NAT,
    FW_GLOBAL_CONFIG_IPSEC_THROUGH_NAT_SERVER_AND_CLIENT_BEHIND_NAT,
    FW_GLOBAL_CONFIG_IPSEC_THROUGH_NAT_MAX
} FW_GLOBAL_CONFIG_IPSEC_THROUGH_NAT_VALUES;

#define FW_GLOBAL_CONFIG_CRL_CHECK_MAX                2
#define FW_GLOBAL_CONFIG_SA_IDLE_TIME_MAX            3600
#define FW_GLOBAL_CONFIG_SA_IDLE_TIME_MIN            300

typedef enum _tag_FW_GLOBAL_CONFIG
{
    FW_GLOBAL_CONFIG_INVALID,
    FW_GLOBAL_CONFIG_POLICY_VERSION_SUPPORTED,
    FW_GLOBAL_CONFIG_CURRENT_PROFILE,
    FW_GLOBAL_CONFIG_DISABLE_STATEFUL_FTP,
    FW_GLOBAL_CONFIG_DISABLE_STATEFUL_PPTP,
    FW_GLOBAL_CONFIG_SA_IDLE_TIME,
    FW_GLOBAL_CONFIG_PRESHARED_KEY_ENCODING,
    FW_GLOBAL_CONFIG_IPSEC_EXEMPT,
    FW_GLOBAL_CONFIG_CRL_CHECK,
    FW_GLOBAL_CONFIG_IPSEC_THROUGH_NAT,
    FW_GLOBAL_CONFIG_POLICY_VERSION,
    FW_GLOBAL_CONFIG_BINARY_VERSION_SUPPORTED,
    FW_GLOBAL_CONFIG_IPSEC_TUNNEL_REMOTE_MACHINE_AUTHORIZATION_LIST,
    FW_GLOBAL_CONFIG_IPSEC_TUNNEL_REMOTE_USER_AUTHORIZATION_LIST,
    FW_GLOBAL_CONFIG_MAX
} FW_GLOBAL_CONFIG;

typedef enum _FW_CONFIG_FLAGS
{
    FW_CONFIG_FLAG_RETURN_DEFAULT_IF_NOT_FOUND = 0x0001
} FW_CONFIG_FLAGS;

typedef struct tag_FW_NETWORK
{
    [string, unique]
    wchar_t* pszName;
    FW_PROFILE_TYPE ProfileType;
} FW_NETWORK, *PFW_NETWORK;

typedef struct tag_FW_ADAPTER
{
    [string, unique]

```

```

    wchar_t* pszFriendlyName;
    GUID Guid;
} FW_ADAPTER, *PFW_ADAPTER;

typedef struct tag_FW_DIAG_APP
{
    [string, unique]
    wchar_t* pszAppPath;
} FW_DIAG_APP, *PFW_DIAG_APP;

typedef
[vl_enum]
enum tag_FW_RULE_CATEGORY
{
    FW_RULE_CATEGORY_BOOT,
    FW_RULE_CATEGORY_STEALTH,
    FW_RULE_CATEGORY_FIREWALL,
    FW_RULE_CATEGORY_CONSEC,
    FW_RULE_CATEGORY_MAX
} FW_RULE_CATEGORY, *PFW_RULE_CATEGORY;

typedef struct tag_FW_PRODUCT
{
    unsigned long dwFlags;
    unsigned long dwNumRuleCategories;
    [size_is(dwNumRuleCategories), unique]
    FW_RULE_CATEGORY* pRuleCategories;
    [string, ref]
    wchar_t* pszDisplayName;
    [string, unique]
    wchar_t* pszPathToSignedProductExe;
} FW_PRODUCT, *PFW_PRODUCT;

typedef enum _tag_FW_IP_VERSION
{
    FW_IP_VERSION_INVALID,
    FW_IP_VERSION_V4,
    FW_IP_VERSION_V6,
    FW_IP_VERSION_MAX
} FW_IP_VERSION;

typedef enum _tag_FW_IPSEC_PHASE
{
    FW_IPSEC_PHASE_INVALID,
    FW_IPSEC_PHASE_1,
    FW_IPSEC_PHASE_2,
    FW_IPSEC_PHASE_MAX
} FW_IPSEC_PHASE;

typedef enum _tag_FW_CS_RULE_FLAGS
{
    FW_CS_RULE_FLAGS_NONE                = 0x00,
    FW_CS_RULE_FLAGS_ACTIVE              = 0x01,
    FW_CS_RULE_FLAGS_DTM                 = 0x02,
    FW_CS_RULE_FLAGS_TUNNEL_BYPASS_IF_ENCRYPTED = 0x08,
    FW_CS_RULE_FLAGS_OUTBOUND_CLEAR     = 0x10,
    FW_CS_RULE_FLAGS_APPLY_AUTHZ        = 0x20,
    FW_CS_RULE_FLAGS_MAX                 = 0x40,
    FW_CS_RULE_FLAGS_MAX_V2_1           = 0x02,

```



```

    FW_CS_RULE_FLAGS_MAX_V2_8                = 0x04
} FW_CS_RULE_FLAGS;

typedef enum _tag_FW_CS_RULE_ACTION
{
    FW_CS_RULE_ACTION_INVALID,
    FW_CS_RULE_ACTION_SECURE_SERVER,
    FW_CS_RULE_ACTION_BOUNDARY,
    FW_CS_RULE_ACTION_SECURE,
    FW_CS_RULE_ACTION_DO_NOT_SECURE,
    FW_CS_RULE_ACTION_MAX
} FW_CS_RULE_ACTION;

typedef struct _tag_FW_CS_RULE2_0
{
    struct _tag_FW_CS_RULE2_0 *pNext;
    unsigned short            wSchemaVersion;
    [string, range(1,512), ref]
    wchar_t*                  wszRuleId;
    [string, range(1,10001)]
    wchar_t*                  wszName;
    [string, range(1,10001)]
    wchar_t*                  wszDescription;
    unsigned long             dwProfiles;
    FW_ADDRESSES              Endpoint1;
    FW_ADDRESSES              Endpoint2;
    FW_INTERFACE_LUIDS        LocalInterfaceIds;
    unsigned long             dwLocalInterfaceTypes;
    unsigned long             dwLocalTunnelEndpointV4;
    unsigned char             LocalTunnelEndpointV6[16];
    unsigned long             dwRemoteTunnelEndpointV4;
    unsigned char             RemoteTunnelEndpointV6[16];
    FW_PORTS                  Endpoint1Ports;
    FW_PORTS                  Endpoint2Ports;
    [range(0,256)]
    unsigned short            wIpProtocol;
    [string, range(1,1001)]
    wchar_t*                  wszPhase1AuthSet;
    [string, range(1,1001)]
    wchar_t*                  wszPhase2CryptoSet;
    [string, range(1,1001)]
    wchar_t*                  wszPhase2AuthSet;
    [range(FW_CS_RULE_ACTION_SECURE_SERVER, FW_CS_RULE_ACTION_MAX)]
    FW_CS_RULE_ACTION         Action;
    unsigned short            wFlags;
    [string, range(1,10001)]
    wchar_t*                  wszEmbeddedContext;
    FW_OS_PLATFORM_LIST        PlatformValidityList;
    [range(FW_RULE_ORIGIN_INVALID, FW_RULE_ORIGIN_MAX-1)]
    FW_RULE_ORIGIN_TYPE        Origin;
    [string, range(1,10001)]
    wchar_t*                  wszGPOName;
    FW_RULE_STATUS             Status;
} FW_CS_RULE2_0, *PFW_CS_RULE2_0;

typedef struct _tag_FW_CS_RULE
{
    struct _tag_FW_CS_RULE *pNext;
    unsigned short            wSchemaVersion;

```

```

[string, range(1,10001), ref]
wchar_t*          wszRuleId;
[string, range(1,10001)]
wchar_t*          wszName;
[string, range(1,10001)]
wchar_t*          wszDescription;
unsigned long     dwProfiles;
FW_ADDRESSES     Endpoint1;
FW_ADDRESSES     Endpoint2;
FW_INTERFACE_LUIDS LocalInterfaceIds;
unsigned long     dwLocalInterfaceTypes;
unsigned long     dwLocalTunnelEndpointV4;
unsigned char     LocalTunnelEndpointV6[16];
unsigned long     dwRemoteTunnelEndpointV4;
unsigned char     RemoteTunnelEndpointV6[16];
FW_PORTS         Endpoint1Ports;
FW_PORTS         Endpoint2Ports;
[range(0,256)]
unsigned short    wIpProtocol;
[string, range(1,10001)]
wchar_t*          wszPhase1AuthSet;
[string, range(1,10001)]
wchar_t*          wszPhase2CryptoSet;
[string, range(1,10001)]
wchar_t*          wszPhase2AuthSet;
[range(FW_CS_RULE_ACTION_SECURE_SERVER, FW_CS_RULE_ACTION_MAX)]
FW_CS_RULE_ACTION Action;
unsigned short    wFlags;
[string, range(1,10001)]
wchar_t*          wszEmbeddedContext;
FW_OS_PLATFORM_LIST PlatformValidityList;
[range(FW_RULE_ORIGIN_INVALID, FW_RULE_ORIGIN_MAX-1)]
FW_RULE_ORIGIN_TYPE Origin;
[string, range(1,10001)]
wchar_t*          wszGPOName;
FW_RULE_STATUS   Status;
[string, range(1,512)]
wchar_t*          wszMMParentRuleId;
unsigned long     Reserved;
[size is((Reserved & FW_OBJECT_CTRL_FLAG_INCLUDE_METADATA) ? 1 : 0)]
PFW_OBJECT_METADATA pMetaData;
} FW_CS_RULE, *PFW_CS_RULE;

typedef enum _tag_FW_AUTH_METHOD
{
    FW_AUTH_METHOD_INVALID,
    FW_AUTH_METHOD_ANONYMOUS,
    FW_AUTH_METHOD_MACHINE_KERB,
    FW_AUTH_METHOD_MACHINE_SHKEY,
    FW_AUTH_METHOD_MACHINE_NTLM,
    FW_AUTH_METHOD_MACHINE_CERT,
    FW_AUTH_METHOD_USER_KERB,
    FW_AUTH_METHOD_USER_CERT,
    FW_AUTH_METHOD_USER_NTLM,
    FW_AUTH_METHOD_MAX
} FW_AUTH_METHOD;

typedef enum _tag_FW_AUTH_SUITE_FLAGS
{

```

```

FW_AUTH_SUITE_FLAGS_NONE = 0x0000,
FW_AUTH_SUITE_FLAGS_CERT_EXCLUDE_CA_NAME = 0x0001,
FW_AUTH_SUITE_FLAGS_HEALTH_CERT = 0x0002,
FW_AUTH_SUITE_FLAGS_PERFORM_CERT_ACCOUNT_MAPPING = 0x0004,
FW_AUTH_SUITE_FLAGS_CERT_SIGNING_ECDSA256 = 0x0008,
FW_AUTH_SUITE_FLAGS_CERT_SIGNING_ECDSA384 = 0x0010,
FW_AUTH_SUITE_FLAGS_MAX_V2_1 = 0x0020,
FW_AUTH_SUITE_FLAGS_INTERMEDIATE_CA = 0x0020,
FW_AUTH_SUITE_FLAGS_MAX = 0x0040
} FW_AUTH_SUITE_FLAGS;

typedef struct _tag_FW_AUTH_SUITE
{
    [range(FW_AUTH_METHOD_INVALID+1, FW_AUTH_METHOD_MAX)]
    FW_AUTH_METHOD Method;
    unsigned short wFlags;
    [switch_type(FW_AUTH_METHOD), switch_is(Method)]
    union
    {
        [case(FW_AUTH_METHOD_MACHINE_CERT,FW_AUTH_METHOD_USER_CERT)]
        struct
        {
            [ref, string]
            wchar_t* wszCAName;
        };

        [case(FW_AUTH_METHOD_MACHINE_SHKEY)]
        struct
        {
            [ref, string]
            wchar_t* wszSHKey;
        };
        [default]
        ;
    };
} FW_AUTH_SUITE, *PFW_AUTH_SUITE;

typedef struct _tag_FW_AUTH_SET
{
    struct _tag_FW_AUTH_SET* pNext;
    unsigned short wSchemaVersion;
    [range(FW_IPSEC_PHASE_INVALID+1, FW_IPSEC_PHASE_MAX-1)]
    FW_IPSEC_PHASE IpSecPhase;
    [string, range(1,255), ref]
    wchar_t* wszSetId;
    [string, range(1,10001)]
    wchar_t* wszName;
    [string, range(1,10001)]
    wchar_t* wszDescription;
    [string, range(1,10001)]
    wchar_t* wszEmbeddedContext;
    [range(0, 1000)]
    unsigned long dwNumSuites;
    [size_is(dwNumSuites)]
    PFW_AUTH_SUITE pSuites;
    [range(FW_RULE_ORIGIN_INVALID, FW_RULE_ORIGIN_MAX-1)]
    FW_RULE_ORIGIN_TYPE Origin;
    [string, range(1,10001)]

```

```

    wchar_t*          wszGPONName;
    FW_RULE_STATUS    Status;
    unsigned long     dwAuthSetFlags;
} FW_AUTH_SET, *PFW_AUTH_SET;

typedef enum _tag_FW_CRYPTOKEY_EXCHANGE_TYPE
{
    FW_CRYPTOKEY_EXCHANGE_NONE = 0,
    FW_CRYPTOKEY_EXCHANGE_DH1,
    FW_CRYPTOKEY_EXCHANGE_DH2,
    FW_CRYPTOKEY_EXCHANGE_ECDH256,
    FW_CRYPTOKEY_EXCHANGE_ECDH384,
    FW_CRYPTOKEY_EXCHANGE_DH2048,
    FW_CRYPTOKEY_EXCHANGE_MAX
} FW_CRYPTOKEY_EXCHANGE_TYPE;

typedef enum _tag_FW_CRYPTO_ENCRYPTION_TYPE
{
    FW_CRYPTO_ENCRYPTION_NONE,
    FW_CRYPTO_ENCRYPTION_DES,
    FW_CRYPTO_ENCRYPTION_3DES,
    FW_CRYPTO_ENCRYPTION_AES128,
    FW_CRYPTO_ENCRYPTION_AES192,
    FW_CRYPTO_ENCRYPTION_AES256,
    FW_CRYPTO_ENCRYPTION_AES_GCM128,
    FW_CRYPTO_ENCRYPTION_AES_GCM192,
    FW_CRYPTO_ENCRYPTION_AES_GCM256,
    FW_CRYPTO_ENCRYPTION_MAX,
    FW_CRYPTO_ENCRYPTION_MAX_V2_0 = FW_CRYPTO_ENCRYPTION_AES_GCM128
} FW_CRYPTO_ENCRYPTION_TYPE;

typedef enum _tag_FW_CRYPTO_HASH_TYPE
{
    FW_CRYPTO_HASH_NONE,
    FW_CRYPTO_HASH_MD5,
    FW_CRYPTO_HASH_SHA1,
    FW_CRYPTO_HASH_SHA256,
    FW_CRYPTO_HASH_SHA384,
    FW_CRYPTO_HASH_AES_GMAC128,
    FW_CRYPTO_HASH_AES_GMAC192,
    FW_CRYPTO_HASH_AES_GMAC256,
    FW_CRYPTO_HASH_MAX,
    FW_CRYPTO_HASH_MAX_V2_0 = FW_CRYPTO_HASH_SHA256
} FW_CRYPTO_HASH_TYPE;

typedef enum _tag_FW_CRYPTO_PROTOCOL_TYPE
{
    FW_CRYPTO_PROTOCOL_INVALID,
    FW_CRYPTO_PROTOCOL_AH,
    FW_CRYPTO_PROTOCOL_ESP,
    FW_CRYPTO_PROTOCOL_BOTH,
    FW_CRYPTO_PROTOCOL_AUTH_NO_ENCAP,
    FW_CRYPTO_PROTOCOL_MAX,
    FW_CRYPTO_PROTOCOL_MAX_2_1 = (FW_CRYPTO_PROTOCOL_BOTH + 1)
} FW_CRYPTO_PROTOCOL_TYPE;

typedef struct _tag_FW_PHASE1_CRYPTOSUITE
{
    [range(FW_CRYPTOKEY_EXCHANGE_NONE, FW_CRYPTOKEY_EXCHANGE_MAX-1)]

```

```

FW_CRYPTO_KEY_EXCHANGE_TYPE KeyExchange;
[range(FW_CRYPTO_ENCRYPTION_NONE+1, FW_CRYPTO_ENCRYPTION_MAX-1)]
FW_CRYPTO_ENCRYPTION_TYPE Encryption;
[range(FW_CRYPTO_HASH_NONE+1, FW_CRYPTO_HASH_MAX-1)]
FW_CRYPTO_HASH_TYPE Hash;
unsigned long dwP1CryptoSuiteFlags;
} FW_PHASE1_CRYPTO_SUITE, *PFW_PHASE1_CRYPTO_SUITE;

typedef struct _tag_FW_PHASE2_CRYPTO_SUITE
{
    [range(FW_CRYPTO_PROTOCOL_INVALID+1, FW_CRYPTO_PROTOCOL_MAX-1)]
    FW_CRYPTO_PROTOCOL_TYPE Protocol;
    FW_CRYPTO_HASH_TYPE AhHash;
    FW_CRYPTO_HASH_TYPE EspHash;
    FW_CRYPTO_ENCRYPTION_TYPE Encryption;
    unsigned long dwTimeoutMinutes;
    unsigned long dwTimeoutKBytes;
    unsigned long dwP2CryptoSuiteFlags;
} FW_PHASE2_CRYPTO_SUITE, *PFW_PHASE2_CRYPTO_SUITE;

typedef enum _tag_FW_PHASE1_CRYPTO_FLAGS
{
    FW_PHASE1_CRYPTO_FLAGS_NONE = 0x00,
    FW_PHASE1_CRYPTO_FLAGS_DO_NOT_SKIP_DH = 0x01,
    FW_PHASE1_CRYPTO_FLAGS_MAX = 0x02
} FW_PHASE1_CRYPTO_FLAGS;

typedef enum _tag_FW_PHASE2_CRYPTO_PFS
{
    FW_PHASE2_CRYPTO_PFS_INVALID,
    FW_PHASE2_CRYPTO_PFS_DISABLE,
    FW_PHASE2_CRYPTO_PFS_PHASE1,
    FW_PHASE2_CRYPTO_PFS_DH1,
    FW_PHASE2_CRYPTO_PFS_DH2,
    FW_PHASE2_CRYPTO_PFS_DH2048,
    FW_PHASE2_CRYPTO_PFS_ECDH256,
    FW_PHASE2_CRYPTO_PFS_ECDH384,
    FW_PHASE2_CRYPTO_PFS_MAX
} FW_PHASE2_CRYPTO_PFS;

typedef struct _tag_FW_CRYPTO_SET
{
    struct _tag_FW_CRYPTO_SET* pNext;
    unsigned short wSchemaVersion;
    [range(FW_IPSEC_PHASE_INVALID+1, FW_IPSEC_PHASE_MAX-1)]
    FW_IPSEC_PHASE IpSecPhase;
    [string, range(1,255), ref]
    wchar_t* wszSetId;
    [string, range(1,10001)]
    wchar_t* wszName;
    [string, range(1,10001)]
    wchar_t* wszDescription;
    [string, range(1,10001)]
    wchar_t* wszEmbeddedContext;
    [switch_type(FW_IPSEC_PHASE), switch_is(IpSecPhase)]
    union
    {
        [case(FW_IPSEC_PHASE_1)]
        struct

```

```

    {
        unsigned short                wFlags;
        [range(0, 1000)]
        unsigned long                 dwNumPhase1Suites;
        [size_is(dwNumPhase1Suites)]
        PFW_PHASE1_CRYPTOSUITE        pPhase1Suites;
        unsigned long                 dwTimeoutMinutes;
        unsigned long                 dwTimeoutSessions;
    };
    [case(FW_IPSEC_PHASE_2)]
    struct
    {
        FW_PHASE2_CRYPTOPFS          Pfs;
        [range(0, 1000)]
        unsigned long                 dwNumPhase2Suites;
        [size_is(dwNumPhase2Suites)]
        PFW_PHASE2_CRYPTOSUITE        pPhase2Suites;
    };
};
[range(FW_RULE_ORIGIN_INVALID, FW_RULE_ORIGIN_MAX-1)]
FW_RULE_ORIGIN_TYPE Origin;
[string, range(1,10001)]
wchar_t*                          wszGPOName;
FW_RULE_STATUS                     Status;
unsigned long                       dwCryptoSetFlags;
} FW_CRYPTOSUITE, *PFW_CRYPTOSUITE;

typedef struct _tag_FW_BYTE_BLOB
{
    [range(0, 10000)]
    unsigned long                     dwSize;
    [size_is(dwSize)]
    unsigned char *                   Blob;
} FW_BYTE_BLOB, *PFW_BYTE_BLOB;

typedef struct _tag_FW_COOKIE_PAIR
{
    unsigned __int64                 Initiator;
    unsigned __int64                 Responder;
} FW_COOKIE_PAIR, *PFW_COOKIE_PAIR;

typedef enum _tag_FW_PHASE1_KEY_MODULE_TYPE
{
    FW_PHASE1_KEY_MODULE_INVALID,
    FW_PHASE1_KEY_MODULE_IKE,
    FW_PHASE1_KEY_MODULE_AUTH_IP,
    FW_PHASE1_KEY_MODULE_MAX
} FW_PHASE1_KEY_MODULE_TYPE;

typedef struct _tag_FW_CERT_INFO
{
    FW_BYTE_BLOB                     SubjectName;
    [range(FW_AUTH_SUITE_FLAGS_NONE, FW_AUTH_SUITE_FLAGS_MAX-1)]
    unsigned long                     dwCertFlags;
} FW_CERT_INFO, *PFW_CERT_INFO;

typedef struct _tag_FW_AUTH_INFO
{
    [range(FW_AUTH_METHOD_INVALID + 1, FW_AUTH_METHOD_MAX)]

```

```

FW_AUTH_METHOD AuthMethod;
[switch_type(FW_AUTH_METHOD), switch_is(AuthMethod)]
union
{
    [case(FW_AUTH_METHOD_MACHINE_CERT,FW_AUTH_METHOD_USER_CERT)]
    struct
    {
        FW_CERT_INFO MyCert;
        FW_CERT_INFO PeerCert;
    };
    [case(FW_AUTH_METHOD_MACHINE_KERB,FW_AUTH_METHOD_USER_KERB)]
    struct
    {
        [string, range(1,10001)]
        wchar_t* wszMyId;
        [string, range(1,10001)]
        wchar_t* wszPeerId;
    };
    [default]
    ;
};
unsigned long dwAuthInfoFlags;
} FW_AUTH_INFO, *PFW_AUTH_INFO;

typedef struct _tag_FW_ENDPOINTS
{
    [range(FW_IP_VERSION_INVALID+1, FW_IP_VERSION_MAX-1)]
    FW_IP_VERSION IpVersion;
    unsigned long dwSourceV4Address;
    unsigned long dwDestinationV4Address;
    unsigned char SourceV6Address[16];
    unsigned char DestinationV6Address[16];
} FW_ENDPOINTS, *PFW_ENDPOINTS;

typedef struct _tag_FW_PHASE1_SA_DETAILS
{
    unsigned __int64 SaId;
    [range(FW_PHASE1_KEY_MODULE_INVALID+1, FW_PHASE1_KEY_MODULE_MAX-1)]
    FW_PHASE1_KEY_MODULE_TYPE KeyModuleType;
    FW_ENDPOINTS Endpoints;
    FW_PHASE1_CRYPTOSUITE SelectedProposal;
    unsigned long dwProposalLifetimeKBytes;
    unsigned long dwProposalLifetimeMinutes;
    unsigned long dwProposalMaxNumPhase2;
    FW_COOKIE_PAIR CookiePair;
    PFW_AUTH_INFO pFirstAuth;
    PFW_AUTH_INFO pSecondAuth;
    unsigned long dwPlSaFlags;
} FW_PHASE1_SA_DETAILS, *PFW_PHASE1_SA_DETAILS;

typedef enum _tag_FW_PHASE2_TRAFFIC_TYPE
{
    FW_PHASE2_TRAFFIC_TYPE_INVALID,
    FW_PHASE2_TRAFFIC_TYPE_TRANSPORT,
    FW_PHASE2_TRAFFIC_TYPE_TUNNEL,
    FW_PHASE2_TRAFFIC_TYPE_MAX
} FW_PHASE2_TRAFFIC_TYPE;

typedef struct _tag_FW_PHASE2_SA_DETAILS

```

```

{
    unsigned __int64          SaId;
    [range(FW_DIR_INVALID+1, FW_DIR_MAX-1)]
    FW_DIRECTION            Direction;
    FW_ENDPOINTS            Endpoints;
    unsigned short          wLocalPort;
    unsigned short          wRemotePort;
    unsigned short          wIpProtocol;
    FW_PHASE2_CRYPTO_SUITE  SelectedProposal;
    FW_PHASE2_CRYPTO_PFS    Pfs;
    GUID                    TransportFilterId;
    unsigned long           dwP2SaFlags;
} FW_PHASE2_SA_DETAILS, *PFW_PHASE2_SA_DETAILS;

typedef
    [switch_type(FW_PROFILE_CONFIG)]
union _FW_PROFILE_CONFIG_VALUE
{
    [case(FW_PROFILE_CONFIG_LOG_FILE_PATH)]
    [string, range(1,10001)]
    wchar_t* wszStr;
    [case(FW_PROFILE_CONFIG_DISABLED_INTERFACES)]
    PFW_INTERFACE_LUIDS pDisabledInterfaces;
    [case(FW_PROFILE_CONFIG_ENABLE_FW,
    FW_PROFILE_CONFIG_DISABLE_STEALTH_MODE,
    FW_PROFILE_CONFIG_SHIELDED,
    FW_PROFILE_CONFIG_DISABLE_UNICAST_RESPONSES_TO_MULTICAST_BROADCAST,
    FW_PROFILE_CONFIG_LOG_DROPPED_PACKETS,
    FW_PROFILE_CONFIG_LOG_SUCCESS_CONNECTIONS,
    FW_PROFILE_CONFIG_LOG_IGNORED_RULES,
    FW_PROFILE_CONFIG_LOG_MAX_FILE_SIZE,
    FW_PROFILE_CONFIG_DISABLE_INBOUND_NOTIFICATIONS,
    FW_PROFILE_CONFIG_AUTH_APPS_ALLOW_USER_PREF_MERGE,
    FW_PROFILE_CONFIG_GLOBAL_PORTS_ALLOW_USER_PREF_MERGE,
    FW_PROFILE_CONFIG_ALLOW_LOCAL_POLICY_MERGE,
    FW_PROFILE_CONFIG_ALLOW_LOCAL_IPSEC_POLICY_MERGE,
    FW_PROFILE_CONFIG_DEFAULT_OUTBOUND_ACTION,
    FW_PROFILE_CONFIG_DEFAULT_INBOUND_ACTION)]
    unsigned long* pdwVal;
} FW_PROFILE_CONFIG_VALUE, *PFW_PROFILE_CONFIG_VALUE;

typedef struct _tag_FW_MM_RULE
{
    struct _tag_FW_MM_RULE *pNext;
    unsigned short wSchemaVersion;
    [string, range(1,512), ref]
    wchar_t* wszRuleId;
    [string, range(1,10001)]
    wchar_t* wszName;
    [string, range(1,10001)]
    wchar_t* wszDescription;
    DWORD dwProfiles;
    FW_ADDRESSES Endpoint1;
    FW_ADDRESSES Endpoint2;
    [string, range(1,255)]
    wchar_t* wszPhase1AuthSet;
    [string, range(1,255)]
    wchar_t* wszPhase1CryptoSet;
    unsigned short wFlags;
}

```



```

    [string, range(1,10001)]
    wchar_t*      wszEmbeddedContext;
    FW_OS_PLATFORM_LIST PlatformValidityList;
    [range(FW_RULE_ORIGIN_INVALID, FW_RULE_ORIGIN_MAX-1)]
    FW_RULE_ORIGIN_TYPE Origin;
    [string, range(1,10001)]
    wchar_t*      wszGPOName;
    FW_RULE_STATUS      Status;
    unsigned long      Reserved;
    [size_is((Reserved & FW_OBJECT_CTRL_FLAG_INCLUDE_METADATA) ? 1 : 0)]
    PFW_OBJECT_METADATA pMetaData;
} FW_MM_RULE, *PFW_MM_RULE;

typedef enum _tag_FW_MATCH_KEY
{
    FW_MATCH_KEY_PROFILE,
    FW_MATCH_KEY_STATUS,
    FW_MATCH_KEY_OBJECTID,
    FW_MATCH_KEY_FILTERID,
    FW_MATCH_KEY_APP_PATH, //The APP Path
    FW_MATCH_KEY_PROTOCOL,
    FW_MATCH_KEY_LOCAL_PORT,
    FW_MATCH_KEY_REMOTE_PORT,
    FW_MATCH_KEY_GROUP,
    FW_MATCH_KEY_SVC_NAME,
    FW_MATCH_KEY_DIRECTION,
    FW_MATCH_KEY_MAX
} FW_MATCH_KEY;

typedef enum _tag_FW_DATA_TYPE
{
    FW_DATA_TYPE_EMPTY,
    FW_DATA_TYPE_UINT8,
    FW_DATA_TYPE_UINT16,
    FW_DATA_TYPE_UINT32,
    FW_DATA_TYPE_UINT64,
    FW_DATA_TYPE_UNICODE_STRING
} FW_DATA_TYPE;

typedef struct _tag_FW_MATCH_VALUE
{
    FW_DATA_TYPE type;
    [switch_type(FW_DATA_TYPE), switch_is(type)]
    union
    {
        [case(FW_DATA_TYPE_UINT8)]
        unsigned char uInt8;
        [case(FW_DATA_TYPE_UINT16)]
        unsigned long uInt16;
        [case(FW_DATA_TYPE_UINT32)]
        unsigned long uInt32;
        [case(FW_DATA_TYPE_UINT64)]
        unsigned __int64 uInt64;
        [case(FW_DATA_TYPE_UNICODE_STRING)]
        struct
        {
            [string, range(1,10001)]
            wchar_t* wszString;
        };
    };
};

```

```

        [case(FW_DATA_TYPE_EMPTY)]
        ;
    };
} FW_MATCH_VALUE;

typedef enum _tag_FW_MATCH_TYPE
{
    FW_MATCH_TYPE_TRAFFIC_MATCH,
    FW_MATCH_TYPE_EQUAL,
    FW_MATCH_TYPE_MAX
} FW_MATCH_TYPE;

typedef struct _tag_FW_QUERY_CONDITION
{
    FW_MATCH_KEY matchKey;
    FW_MATCH_TYPE matchType;
    FW_MATCH_VALUE matchValue;
} FW_QUERY_CONDITION, *PFW_QUERY_CONDITION;

typedef struct _tag_FW_QUERY_CONDITIONS
{
    unsigned long dwNumEntries;
    [size_is(dwNumEntries)]
    FW_QUERY_CONDITION *AandedConditions;
} FW_QUERY_CONDITIONS, *PFW_QUERY_CONDITIONS;

typedef struct _tag_FW_QUERY
{
    unsigned short wSchemaVersion;
    unsigned long dwNumEntries;
    [size_is(dwNumEntries)]
    FW_QUERY_CONDITIONS *ORConditions;
    FW_RULE_STATUS Status;
} FW_QUERY, *PFW_QUERY;

cpp_quote("#endif // __FIREWALL_H")
cpp_quote("#define MIDL_user_allocate MIDL_fw_allocate")
cpp_quote("#define MIDL_user_free MIDL_fw_free")
cpp_quote("void * __RPC_USER MIDL_fw_allocate(size_t size);")
cpp_quote("void __RPC_USER MIDL_fw_free(void * );")

[
    uuid(6b5bddd1-528c-422c-af8c-a4079be4fe48),
    version(1.0),
    pointer_default(unique)
]
interface RemoteFW
{
    typedef
    handle_t FW_CONN_HANDLE;

    typedef
    [context_handle]
    void* FW_POLICY_STORE_HANDLE;

    typedef
    [ref]

```

```

FW_POLICY_STORE_HANDLE *PFW_POLICY_STORE_HANDLE;

typedef
[context_handle]
void* FW_PRODUCT_HANDLE;

unsigned long
RRPC_FWOpenPolicyStore(
[in] FW_CONN_HANDLE          rpcConnHandle,
[in] unsigned short          BinaryVersion,
[in, range(FW_STORE_TYPE_INVALID+1, FW_STORE_TYPE_MAX-1)]
FW_STORE_TYPE                StoreType,
[in, range(FW_POLICY_ACCESS_RIGHT_INVALID+1, FW_POLICY_ACCESS_RIGHT_MAX-1)]
FW_POLICY_ACCESS_RIGHT AccessRight,
[in] unsigned long           dwFlags,
[out] PFW_POLICY_STORE_HANDLE phPolicyStore
);

unsigned long
RRPC_FWClosePolicyStore(
[in] FW_CONN_HANDLE          rpcConnHandle,
[in, out] PFW_POLICY_STORE_HANDLE phPolicyStore
);

unsigned long
RRPC_FWRestoreDefaults(
[in] FW_CONN_HANDLE rpcConnHandle
);

unsigned long
RRPC_FWGetGlobalConfig(
[in] FW_CONN_HANDLE          rpcConnHandle,
[in] unsigned short          BinaryVersion,
[in] FW_STORE_TYPE          StoreType,
[in, range(FW_GLOBAL_CONFIG_INVALID+1, FW_GLOBAL_CONFIG_MAX-1)]
FW_GLOBAL_CONFIG            configID,
[in] unsigned long           dwFlags,
[in, out, unique, size_is(cbData), length_is(*pcbTransmittedLen)]
unsigned char*               pBuffer,
[in] unsigned long           cbData,
[in,out] unsigned long       *pcbTransmittedLen,
[out] unsigned long          *pcbRequired
);

unsigned long
RRPC_FWSetGlobalConfig(
[in] FW_CONN_HANDLE          rpcConnHandle,
[in] unsigned short          BinaryVersion,
[in] FW_STORE_TYPE          StoreType,
[in, range(FW_GLOBAL_CONFIG_INVALID+1, FW_GLOBAL_CONFIG_MAX-1)]
FW_GLOBAL_CONFIG            configID,
[in, unique, size_is(dwBufSize)]
unsigned char *lpBuffer,
[in, range(0, 10*1024)] unsigned long dwBufSize
);

unsigned long
RRPC_FWAddFirewallRule(
[in] FW_CONN_HANDLE          rpcConnHandle,

```

```

[in] FW_POLICY_STORE_HANDLE hPolicyStore,
[in] PFW_RULE2_0           pRule
);

unsigned long
RRPC_FWSetFirewallRule(
[in] FW_CONN_HANDLE       rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE hPolicyStore,
[in] PFW_RULE2_0         pRule
);

unsigned long
RRPC_FWDeleteFirewallRule(
[in] FW_CONN_HANDLE       rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE hPolicyStore,
[in, string, ref] const wchar_t* wszRuleID
);

unsigned long
RRPC_FWDeleteAllFirewallRules(
[in] FW_CONN_HANDLE       rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE hPolicyStore
);

unsigned long
RRPC_FWEnumFirewallRules(
[in] FW_CONN_HANDLE       rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE hPolicyStore,
[in] unsigned long        dwFilteredByStatus,
[in] unsigned long        dwProfileFilter,
[in] unsigned short       wFlags,
[out, ref] unsigned long  *pdwNumRules,
[out] PFW_RULE2_0        *ppRules
);

unsigned long
RRPC_FWGetConfig(
[in] FW_CONN_HANDLE       rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE hPolicyStore,
[in, range(FW_PROFILE_CONFIG_ENABLE_FW, FW_PROFILE_CONFIG_MAX-1)]
FW_PROFILE_CONFIG        configID,
[in] FW_PROFILE_TYPE      Profile,
[in] unsigned long        dwFlags,
[in, out, unique, size_is(cbData), length_is(*pcbTransmittedLen)]
unsigned char*           pBuffer,
[in] unsigned long        cbData,
[in,out] unsigned long    *pcbTransmittedLen,
[out] unsigned long       *pcbRequired
);

unsigned long
RRPC_FWSetConfig(
[in] FW_CONN_HANDLE       rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE hPolicyStore,
[in, range(FW_PROFILE_CONFIG_ENABLE_FW, FW_PROFILE_CONFIG_MAX-1)]
FW_PROFILE_CONFIG        configID,
[in] FW_PROFILE_TYPE      Profile,
[in, switch_is(configID)] FW_PROFILE_CONFIG_VALUE pConfig,
[in, range(0, 10*1024)] unsigned long dwBufSize
);

```

```

);

unsigned long
    RRPC_FWAddConnectionSecurityRule(
        [in] FW_CONN_HANDLE          rpcConnHandle,
        [in] FW_POLICY_STORE_HANDLE  hPolicy,
        [in] PFW_CS_RULE2_0          pRule
    );

unsigned long
    RRPC_FWSetConnectionSecurityRule(
        [in] FW_CONN_HANDLE          rpcConnHandle,
        [in] FW_POLICY_STORE_HANDLE  hPolicy,
        [in] PFW_CS_RULE2_0          pRule
    );

unsigned long
    RRPC_FWDeleteConnectionSecurityRule(
        [in] FW_CONN_HANDLE          rpcConnHandle,
        [in] FW_POLICY_STORE_HANDLE  hPolicy,
        [in, string, ref] wchar_t    *pRuleId
    );

unsigned long
    RRPC_FWDeleteAllConnectionSecurityRules(
        [in] FW_CONN_HANDLE          rpcConnHandle,
        [in] FW_POLICY_STORE_HANDLE  hPolicy
    );

unsigned long
    RRPC_FWEnumConnectionSecurityRules(
        [in] FW_CONN_HANDLE          rpcConnHandle,
        [in] FW_POLICY_STORE_HANDLE  hPolicy,
        [in] unsigned long           dwFilteredByStatus,
        [in] unsigned long           dwProfileFilter,
        [in] unsigned short          wFlags,
        [out, ref] unsigned long *    pdwNumRules,
        [out] PFW_CS_RULE2_0*        ppRules
    );

unsigned long
    RRPC_FWAddAuthenticationSet(
        [in] FW_CONN_HANDLE          rpcConnHandle,
        [in] FW_POLICY_STORE_HANDLE  hPolicy,
        [in] PFW_AUTH_SET            pAuth
    );

unsigned long
    RRPC_FWSetAuthenticationSet(
        [in] FW_CONN_HANDLE          rpcConnHandle,
        [in] FW_POLICY_STORE_HANDLE  hPolicy,
        [in] PFW_AUTH_SET            pAuth
    );

unsigned long
    RRPC_FWDeleteAuthenticationSet(
        [in] FW_CONN_HANDLE          rpcConnHandle,
        [in] FW_POLICY_STORE_HANDLE  hPolicy,
        [in, range(FW_IPSEC_PHASE_INVALID+1, FW_IPSEC_PHASE_MAX-1)]

```

```

        FW_IPSEC_PHASE            IpSecPhase,
[in, string, ref] const wchar_t* wszSetId
    );

unsigned long
    RRPC_FWDeleteAllAuthenticationSets(
[in] FW_CONN_HANDLE            rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE    hPolicy,
[in, range(FW_IPSEC_PHASE_INVALID+1, FW_IPSEC_PHASE_MAX-1)]
    FW_IPSEC_PHASE            IpSecPhase
    );

unsigned long
    RRPC_FWEnumAuthenticationSets(
[in] FW_CONN_HANDLE            rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE    hPolicy,
[in, range(FW_IPSEC_PHASE_INVALID+1, FW_IPSEC_PHASE_MAX-1)]
    FW_IPSEC_PHASE            IpSecPhase,
[in] unsigned long            dwFilteredByStatus,
[in] unsigned short            wFlags,
[out] unsigned long*          pdwNumAuthSets,
[out] PFW_AUTH_SET*          ppAuth
    );

unsigned long
    RRPC_FWAddCryptoSet(
[in] FW_CONN_HANDLE            rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE    hPolicy,
[in] PFW_CRYPTTO_SET          pCrypto
    );

unsigned long
    RRPC_FWSetCryptoSet(
[in] FW_CONN_HANDLE            rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE    hPolicy,
[in] PFW_CRYPTTO_SET          pCrypto
    );

unsigned long
    RRPC_FWDeleteCryptoSet(
[in] FW_CONN_HANDLE            rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE    hPolicy,
[in, range(FW_IPSEC_PHASE_INVALID+1, FW_IPSEC_PHASE_MAX-1)]
    FW_IPSEC_PHASE            IpSecPhase,
[in, string, ref] const wchar_t* wszSetId
    );

unsigned long
    RRPC_FWDeleteAllCryptoSets(
[in] FW_CONN_HANDLE            rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE    hPolicy,
[in, range(FW_IPSEC_PHASE_INVALID+1, FW_IPSEC_PHASE_MAX-1)]
    FW_IPSEC_PHASE            IpSecPhase
    );

unsigned long
    RRPC_FWEnumCryptoSets(
[in] FW_CONN_HANDLE            rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE    hPolicy,

```

```

[in, range(FW_IPSEC_PHASE_INVALID+1, FW_IPSEC_PHASE_MAX-1)]
    FW_IPSEC_PHASE                IpSecPhase,
[in] unsigned long                 dwFilteredByStatus,
[in] unsigned short                wFlags,
[out, ref] unsigned long*         pdwNumSets,
[out] PFW_CRYPTOSSET*            ppCryptoSets
);

unsigned long
    RRPC_FWEnumPhase1SAs(
[in] FW_CONN_HANDLE               rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE       hPolicy,
[in,unique] PFW_ENDPOINTS         pEndpoints,
[out,ref] unsigned long*         pdwNumSAs,
[out,size_is( , *pdwNumSAs)] PFW_PHASE1_SA_DETAILS* ppSAs
);

unsigned long
    RRPC_FWEnumPhase2SAs(
[in] FW_CONN_HANDLE               rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE       hPolicy,
[in, unique] PFW_ENDPOINTS         pEndpoints,
[out,ref] unsigned long*         pdwNumSAs,
[out,size_is( , *pdwNumSAs)] PFW_PHASE2_SA_DETAILS* ppSAs
);

unsigned long
    RRPC_FWDeletePhase1SAs(
[in] FW_CONN_HANDLE               rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE       hPolicy,
[in, unique] PFW_ENDPOINTS         pEndpoints
);

unsigned long
    RRPC_FWDeletePhase2SAs(
[in] FW_CONN_HANDLE               rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE       hPolicy,
[in, unique] PFW_ENDPOINTS         pEndpoints
);

unsigned long
    RRPC_FWEnumProducts(
[in] FW_CONN_HANDLE rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE hPolicyStore,
[out] unsigned long* pdwNumProducts,
[out, size_is(*pdwNumProducts)] PFW_PRODUCT* ppProducts
);

unsigned long
RRPC_FWAddMainModeRule(
[in] FW_CONN_HANDLE               rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE       hPolicy,
[in] PFW_MM_RULE                   pMMRule,
[out] FW_RULE_STATUS *            pStatus
);

unsigned long
RRPC_FWSetMainModeRule(

```

```

[in] FW_CONN_HANDLE         rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE hPolicy,
[in] PFW_MM_RULE            pMMRule,
[out] FW_RULE_STATUS *     pStatus
);

unsigned long
RRPC_FWDeleteMainModeRule(
[in] FW_CONN_HANDLE         rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE hPolicy,
[in, string, ref] LPWSTR   pRuleId
);

unsigned long
RRPC_FWDeleteAllMainModeRules(
[in] FW_CONN_HANDLE         rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE hPolicy
);

unsigned long
RRPC_FWEnumMainModeRules(
[in] FW_CONN_HANDLE         rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE hPolicy,
[in] unsigned long         dwFilteredByStatus,
[in] unsigned long         dwProfileFilter,
[in] unsigned short        wFlags,
[out, ref] unsigned long   *pdwNumRules,
[out] PFW_MM_RULE *       ppMMRules
);

unsigned long
RRPC_FWQueryFirewallRules(
[in] FW_CONN_HANDLE         rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE hPolicy,
[in] PFW_QUERY              pQuery,
[in] unsigned short         wFlags,
[out, ref] unsigned long   *pdwNumRules,
[out] PFW_RULE *           ppRules
);

unsigned long
RRPC_FWQueryConnectionSecurityRules(
[in] FW_CONN_HANDLE         rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE hPolicy,
[in] PFW_QUERY              pQuery,
[in] unsigned short         wFlags,
[out, ref] unsigned long   *pdwNumRules,
[out] PFW_CS_RULE *        ppRules
);

unsigned long
RRPC_FWQueryMainModeRules(
[in] FW_CONN_HANDLE         rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE hPolicy,
[in] PFW_QUERY              pQuery,
[in] unsigned short         wFlags,
[out, ref] unsigned long   *pdwNumRules,
[out] PFW_MM_RULE *       ppMMRules
);

```



```

unsigned long
RRPC_FWQueryAuthenticationSets(
    [in] FW_CONN_HANDLE      rpcConnHandle,
    [in] FW_POLICY_STORE_HANDLE hPolicy,
    [in, range(FW_IPSEC_PHASE_INVALID+1, FW_IPSEC_PHASE_MAX-1)]
    FW_IPSEC_PHASE          IPsecPhase,
    [in] PFW_QUERY          pQuery,
    [in] unsigned short      wFlags,
    [out, ref] unsigned long *pdwNumSets,
    [out] PFW_AUTH_SET *      ppAuthSets
);

unsigned long
RRPC_FWQueryCryptoSets(
    [in] FW_CONN_HANDLE      rpcConnHandle,
    [in] FW_POLICY_STORE_HANDLE hPolicy,
    [in, range(FW_IPSEC_PHASE_INVALID+1, FW_IPSEC_PHASE_MAX-1)]
    FW_IPSEC_PHASE          IPsecPhase,
    [in] PFW_QUERY          pQuery,
    [in] unsigned short      wFlags,
    [out, ref] unsigned long *pdwNumSets,
    [out] PFW_CRYPTTO_SET *  ppCryptoSets
);

unsigned long
RRPC_FWEnumNetworks(
    [in] FW_CONN_HANDLE rpcConnHandle,
    [in] FW_POLICY_STORE_HANDLE hPolicyStore,
    [out] unsigned long *pdwNumNetworks,
    [out, size_is(*pdwNumNetworks)] PFW_NETWORK* ppNetworks
);

unsigned long
RRPC_FWEnumAdapters(
    [in] FW_CONN_HANDLE rpcConnHandle,
    [in] FW_POLICY_STORE_HANDLE hPolicyStore,
    [out] unsigned long *pdwNumAdapters,
    [out, size_is(*pdwNumAdapters)] PFW_ADAPTER* ppAdapters
);

unsigned long
RRPC_FWGetGlobalConfig2_10(
    [in] FW_CONN_HANDLE      rpcConnHandle,
    [in] unsigned short      BinaryVersion,
    [in] FW_STORE_TYPE       StoreType,
    [in, range(FW_GLOBAL_CONFIG_INVALID+1, FW_GLOBAL_CONFIG_MAX-1)]
    FW_GLOBAL_CONFIG         configID,
    [in] unsigned long        dwFlags,
    [in, out, unique, size_is(cbData), length_is(*pcbTransmittedLen)]
    BYTE*                     pBuffer,
    [in] unsigned long        cbData,
    [in,out] unsigned long    *pcbTransmittedLen,
    [out] unsigned long        *pcbRequired,
    [out] FW_RULE_ORIGIN_TYPE * pOrigin
);

unsigned long
RRPC_FWGetConfig2_10(

```

```

[in] FW_CONN_HANDLE          rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE  hPolicyStore,
[in, range(FW_PROFILE_CONFIG_ENABLE_FW, FW_PROFILE_CONFIG_MAX-1)]
    FW_PROFILE_CONFIG        configID,
[in] FW_PROFILE_TYPE         Profile,
[in] unsigned long           dwFlags,
[in, out, unique, size_is(cbData), length_is(*pcbTransmittedLen)]
    BYTE*                     pBuffer,
[in] unsigned long           cbData,
[in,out] unsigned long       *pcbTransmittedLen,
[out] unsigned long          *pcbRequired,
[out] FW_RULE_ORIGIN_TYPE * pOrigin
);

unsigned long
RRPC_FWAddFirewallRule2_10(
[in] FW_CONN_HANDLE          rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE  hPolicyStore,
[in] PFW_RULE                pRule,
[out] FW_RULE_STATUS *      pStatus
);

unsigned long
RRPC_FWSetFirewallRule2_10(
[in] FW_CONN_HANDLE          rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE  hPolicyStore,
[in] PFW_RULE                pRule,
[out] FW_RULE_STATUS *      pStatus
);

unsigned long
RRPC_FWEnumFirewallRules2_10(
[in] FW_CONN_HANDLE          rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE  hPolicyStore,
[in] unsigned long           dwFilteredByStatus,
[in] unsigned long           dwProfileFilter,
[in] unsigned short          wFlags,
[out, ref] unsigned long     *pdwNumRules,
[out] PFW_RULE              *ppRules
);

unsigned long
RRPC_FWAddConnectionSecurityRule2_10(
[in] FW_CONN_HANDLE          rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE  hPolicy,
[in] PFW_CS_RULE            pRule,
[out] FW_RULE_STATUS *      pStatus
);

unsigned long
RRPC_FWSetConnectionSecurityRule2_10(
[in] FW_CONN_HANDLE          rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE  hPolicy,
[in] PFW_CS_RULE            pRule,
[out] FW_RULE_STATUS *      pStatus
);

unsigned long
RRPC_FWEnumConnectionSecurityRules2_10(

```

```

[in] FW_CONN_HANDLE      rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE hPolicyStore,
[in] unsigned long      dwFilteredByStatus,
[in] unsigned long      dwProfileFilter,
[in] unsigned short     wFlags,
[out, ref] unsigned long *pdwNumRules,
[out] PFW_CS_RULE      *ppRules
);

unsigned long
RRPC_FWAddAuthenticationSet2_10(
[in] FW_CONN_HANDLE      rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE hPolicy,
[in] PFW_AUTH_SET       pAuth,
[out] FW_RULE_STATUS * pStatus
);

unsigned long
RRPC_FWSetAuthenticationSet2_10(
[in] FW_CONN_HANDLE      rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE hPolicy,
[in] PFW_AUTH_SET       pAuth,
[out] FW_RULE_STATUS * pStatus
);

unsigned long
RRPC_FWEnumAuthenticationSets2_10(
[in] FW_CONN_HANDLE      rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE hPolicyStore,
[in, range(FW_IPSEC_PHASE_INVALID+1, FW_IPSEC_PHASE_MAX-1)]
    FW_IPSEC_PHASE IpSecPhase,
[in] unsigned long      dwFilteredByStatus,
[in] unsigned short     wFlags,
[out, ref] unsigned long *pdwNumAuthSets,
[out] PFW_AUTH_SET      *ppAuth
);

unsigned long
RRPC_FWAddCryptoSet2_10(
[in] FW_CONN_HANDLE      rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE hPolicy,
[in] PFW_CRYPTTO_SET     pCrypto,
[out] FW_RULE_STATUS * pStatus
);

unsigned long
RRPC_FWSetCryptoSet2_10(
[in] FW_CONN_HANDLE      rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE hPolicy,
[in] PFW_CRYPTTO_SET     pCrypto,
[out] FW_RULE_STATUS * pStatus
);

unsigned long
RRPC_FWEnumCryptoSets2_10(
[in] FW_CONN_HANDLE      rpcConnHandle,
[in] FW_POLICY_STORE_HANDLE hPolicyStore,
[in, range(FW_IPSEC_PHASE_INVALID+1, FW_IPSEC_PHASE_MAX-1)]
    FW_IPSEC_PHASE IpSecPhase,

```

```
[in] unsigned long      dwFilteredByStatus,  
[in] unsigned short    wFlags,  
[out, ref] unsigned long *pdwNumSets,  
[out] PFW_AUTH_SET     *ppCryptoSets  
);  
  
}
```

7 Appendix B: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Windows Vista® operating system
- Windows Server® 2008 operating system
- Windows® 7 operating system
- Windows Server® 2008 R2 operating system

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

[<1> Section 1.7:](#) Windows Vista uses version 0x0200. Windows Vista SP1 and Windows Server 2008 use version 0x0201. Windows 7 and Windows Server 2008 R2 use version 0x020A.

[<2> Section 2.2.6:](#) For Windows Vista SP1, Windows Server 2008, Windows 7, and Windows Server 2008 R2, unspecified addresses are allowed. Unspecified addresses are also allowed on Windows Vista if the Security Update for Windows Vista specified in [\[MSKB-935807\]](#) is applied.

[<3> Section 2.2.29:](#) Windows uses the three fields of the **FW_OS_PLATFORM** data type to identify Windows platform types. The fields in this data type correspond to the fields of the Windows OSVERSIONINFOEX data type (for more information, see [\[MSDN-OSVERSIONINFOEX\]](#)). The **bPlatform** field in this specification corresponds to the **dwPlatformId** field in MSDN. The **bMajorVersion** field in this specification corresponds to the **dwMajorVersion** field in MSDN. The **bMinorVersion** field in this specification corresponds to the **dwMinorVersion** field in MSDN. The Windows firewall and advanced security components extract the OSVERSIONINFOEX values and use them to enforce PlatformValidityList conditions in [FW_RULE \(section 2.2.36\)](#) and [FW_CS_RULE \(section 2.2.53\)](#) rules.

[<4> Section 2.2.31:](#) The Windows Vista firewall and advanced security component has specific hardcoded objects for default Phase 1 and Phase 2 authentication (see section [2.2.57](#)) and cryptographic (see section [2.2.66](#)) sets, which are used when no other default sets are specified in any store. These sets are hardcoded as follows:

```
#define FW_DEFAULT_P1_AUTH_SET_NAME_STR
    L"Service Hardcoded Default Phase1 AuthSet"
#define FW_DEFAULT_P2_AUTH_SET_NAME_STR
    L"Service Hardcoded Default Phase1 AuthSet"

FW_AUTH_SUITE g_DefaultAuthSuitePhase1[] =
{
    { FW_AUTH_METHOD_MACHINE_KERB, {0} }
};
FW_AUTH_SET g_DefaultAuthSetPhase1 =
```

```

{
    NULL,
    0x0200,
    FW_IPSEC_PHASE_1,
    FW_DEFAULT_PHASE1_AUTH_SET,
    FW_DEFAULT_P1_AUTH_SET_NAME_STR,
    FW_DEFAULT_P1_AUTH_SET_NAME_STR,
    NULL,
    RTL_NUMBER_OF(g_DefaultAuthSuitePhase1),
    g_DefaultAuthSuitePhase1,
    FW_RULE_ORIGIN_HARDCODED,
    NULL,
    FW_RULE_STATUS_OK
};

FW_AUTH_SET g_DefaultAuthSetPhase2 =
{
    NULL,
    0x0200,
    FW_IPSEC_PHASE_2,
    FW_DEFAULT_PHASE2_AUTH_SET,
    FW_DEFAULT_P2_AUTH_SET_NAME_STR,
    FW_DEFAULT_P2_AUTH_SET_NAME_STR,
    NULL,
    0,
    NULL,
    FW_RULE_ORIGIN_HARDCODED,
    NULL,
    FW_RULE_STATUS_OK
};

#define FW_DEFAULT_P1_CRYPTOSUITE_NAME_STR
    L"Service Hardcoded Default Phase1 CryptoSet"
#define FW_DEFAULT_P2_CRYPTOSUITE_NAME_STR
    L"Service Hardcoded Default Phase2 CryptoSet"

FW_PHASE1_CRYPTOSUITE g_DefaultCryptoSuitesPhase1[] =
{
    {FW_CRYPTOSUITE_KEY_EXCHANGE_DH2,
     FW_CRYPTOSUITE_ENCRYPTION_AES128,
     FW_CRYPTOSUITE_HASH_SHA1},
    {FW_CRYPTOSUITE_KEY_EXCHANGE_DH2,
     FW_CRYPTOSUITE_ENCRYPTION_3DES,
     FW_CRYPTOSUITE_HASH_SHA1}
};

FW_CRYPTOSUITE g_DefaultCryptoSetPhase1 =
{
    NULL,
    0x0200,
    FW_IPSEC_PHASE_1,
    FW_PHASE1_CRYPTOSUITE, // Notice: it should be called default
                          // phase1 crypto set
    FW_DEFAULT_P1_CRYPTOSUITE_NAME_STR,
    FW_DEFAULT_P1_CRYPTOSUITE_NAME_STR,
    NULL,
    {
        0, //flags
        0, //RTL_NUMBER_OF(g_DefaultCryptoSuitesP1),
    }
};

```

```

        0, //g_DefaultCryptoSuitesP1,
        0, //480,
        0
    },
    FW_RULE_ORIGIN_HARDCODED, // Notice: it should be set to rule
                             // origin service defined
    NULL,
    FW_RULE_STATUS_OK
};

FW_PHASE2_CRYPTO_SUITE g_DefaultCryptoSuitesPhase2[] =
{
    {FW_CRYPTO_PROTOCOL_ESP,
     FW_CRYPTO_HASH_NONE,
     FW_CRYPTO_HASH_SHA1,
     FW_CRYPTO_ENCRYPTION_NONE,
     FW_DEFAULT_CRYPTO_PHASE2_TIMEOUT_MINUTES,
     FW_DEFAULT_CRYPTO_PHASE2_TIMEOUT_KBYTES},
    {FW_CRYPTO_PROTOCOL_ESP,
     FW_CRYPTO_HASH_NONE,
     FW_CRYPTO_HASH_SHA1,
     FW_CRYPTO_ENCRYPTION_AES128,
     FW_DEFAULT_CRYPTO_PHASE2_TIMEOUT_MINUTES,
     FW_DEFAULT_CRYPTO_PHASE2_TIMEOUT_KBYTES},
    {FW_CRYPTO_PROTOCOL_ESP,
     FW_CRYPTO_HASH_NONE,
     FW_CRYPTO_HASH_SHA1,
     FW_CRYPTO_ENCRYPTION_3DES,
     FW_DEFAULT_CRYPTO_PHASE2_TIMEOUT_MINUTES,
     FW_DEFAULT_CRYPTO_PHASE2_TIMEOUT_KBYTES},
    {FW_CRYPTO_PROTOCOL_AH,
     FW_CRYPTO_HASH_SHA1,
     FW_CRYPTO_HASH_NONE,
     FW_CRYPTO_ENCRYPTION_NONE,
     FW_DEFAULT_CRYPTO_PHASE2_TIMEOUT_MINUTES,
     FW_DEFAULT_CRYPTO_PHASE2_TIMEOUT_KBYTES}
};

FW_CRYPTO_SET g_DefaultCryptoSetPhase2 =
{
    NULL,
    0x0200,
    FW_IPSEC_PHASE_2,
    FW_DEFAULT_PHASE2_CRYPTO_SET,
    FW_DEFAULT_P2_CRYPTO_SET_NAME_STR,
    FW_DEFAULT_P2_CRYPTO_SET_NAME_STR,
    NULL,
    {
        {
            0, // FW_PHASE2_CRYPTO_PFS_DISABLE, //PFS
            0, // RTL_NUMBER_OF(g_DefaultCryptoSuitesP2),
            0, // g_DefaultCryptoSuitesP2
        }
    },
    FW_RULE_ORIGIN_HARDCODED, // Notice: it should be set to rule
                             // origin service defined
    NULL,
    FW_RULE_STATUS_OK
}

```

```

};

void FwHardCodedDefaultSetsInit()
{
    // Init Phase 1 Crypto.
    g_DefaultCryptoSetPhase1.dwNumPhase1Suites =
        RTL_NUMBER_OF(g_DefaultCryptoSuitesPhase1);
    g_DefaultCryptoSetPhase1.pPhase1Suites =
        g_DefaultCryptoSuitesPhase1;
    g_DefaultCryptoSetPhase1.dwTimeOutMinutes = 480;

    //Init Phase 2 Crypto
    g_DefaultCryptoSetPhase2.Pfs =
        FW_PHASE2_CRYPTOPFS_DISABLE;
    g_DefaultCryptoSetPhase2.dwNumPhase2Suites =
        RTL_NUMBER_OF(g_DefaultCryptoSuitesPhase2);
    g_DefaultCryptoSetPhase2.pPhase2Suites =
        g_DefaultCryptoSuitesPhase2;
}

```

Note that `RTL_NUMBER_OF` is a standard macro, equivalent to `sizeof()` (see [\[MSDN-C++NUMS/OPERS\]](#)).

<5> [Section 2.2.32](#): Windows Vista selects enforcement state objects to be filter handles provided by the Windows Filtering Platform (for more information, see [\[MSWFPSDK\]](#)).

<6> [Section 2.2.34](#): Windows 7 and Windows Server 2008 R2 allow applications to request NAT traversal traffic by using the `IPV6_PROTECTION_LEVEL` API. For more information, see [\[MSDN-IPV6PROTLEVEL\]](#).

<7> [Section 2.2.36](#): Rules with **wSchemaVersion** less than 0x000200 but greater than or equal to 0x000100 are not allowed to be written to the local store.

<8> [Section 2.2.36](#): On Windows 7 and Windows Server 2008 R2 the **wszRuleId** size cannot be greater than or equal to 512 characters. On Windows Vista and Windows Server 2008 it cannot be greater than or equal to 1000 characters.

<9> [Section 2.2.42](#): Windows Vista selects a default value for the profile configuration options and the global configurations options. These configurations default values are secure, and it is recommended to use these values as default values. Profile configuration options default values:

```

FW_PROFILE_CONFIG_ENABLE_FW .- TRUE.
FW_PROFILE_CONFIG_DISABLE_STEALTH_MODE .- FALSE.
FW_PROFILE_CONFIG_SHIELDED .- FALSE.
FW_PROFILE_CONFIG_DISABLE_UNICAST_RESPONSES_TO_MULTICAST_BROADCAST
    .- FALSE.
FW_PROFILE_CONFIG_LOG_DROPPED_PACKETS .- FALSE.
FW_PROFILE_CONFIG_LOG_SUCCESS_CONNECTIONS .- FALSE.
FW_PROFILE_CONFIG_LOG_IGNORED_RULES .- TRUE.
FW_PROFILE_CONFIG_LOG_MAX_FILE_SIZE .- 1024.
FW_PROFILE_CONFIG_LOG_FILE_PATH .- L"".
FW_PROFILE_CONFIG_DISABLE_INBOUND_NOTIFICATIONS .- FALSE.
FW_PROFILE_CONFIG_AUTH_APPS_ALLOW_USER_PREF_MERGE .- TRUE.
FW_PROFILE_CONFIG_GLOBAL_PORTS_ALLOW_USER_PREF_MERGE .- TRUE.
FW_PROFILE_CONFIG_ALLOW_LOCAL_POLICY_MERGE .- TRUE.

```



```
FW_PROFILE_CONFIG_ALLOW_LOCAL_IPSEC_POLICY_MERGE .- TRUE.
FW_PROFILE_CONFIG_DISABLED_INTERFACES .- {0}.
FW_PROFILE_CONFIG_DEFAULT_OUTBOUND_ACTION .- 0 (0 is allow).
FW_PROFILE_CONFIG_DEFAULT_INBOUND_ACTION.- 1 (1 is block).
```

Global configuration options default values:

```
FW_GLOBAL_CONFIG_POLICY_VERSION_SUPPORTED .- 0x0200
on Windows Vista.
FW_GLOBAL_CONFIG_POLICY_VERSION_SUPPORTED .- 0x0201
on Windows Vista SP1 and Windows Server 2008.
FW_GLOBAL_CONFIG_CURRENT_PROFILE .- FW_PROFILE_TYPE_PUBLIC.
FW_GLOBAL_CONFIG_DISABLE_STATEFUL_FTP .- FALSE.
FW_GLOBAL_CONFIG_DISABLE_STATEFUL_PPTP .- FALSE.
FW_GLOBAL_CONFIG_SA_IDLE_TIME .- 300.
FW_GLOBAL_CONFIG_PRESHARED_KEY_ENCODING
    .- FW_GLOBAL_CONFIG_PRESHARED_KEY_ENCODING_UTF_8.
FW_GLOBAL_CONFIG_IPSEC_EXEMPT
    .- FW_GLOBAL_CONFIG_IPSEC_EXEMPT_NEIGHBOR_DISC.
FW_GLOBAL_CONFIG_CRL_CHECK .- 0.
FW_GLOBAL_CONFIG_IPSEC_THROUGH_NAT
    .- FW_GLOBAL_CONFIG_IPSEC_THROUGH_NAT_SERVER_BEHIND_NAT.
FW_GLOBAL_CONFIG_POLICY_VERSION .- 0x0200.
FW_GLOBAL_CONFIG_BINARY_VERSION_SUPPORTED .- 0x201. This value is
present only in Windows Vista SP1 and Windows Server 2008.
```

[<10> Section 2.2.53:](#) On Windows 7 and Windows Server 2008 R2 the **wszRuleId** size is less than 512 characters. On Windows Vista and Windows Server 2008 it is less than 1000 characters.

[<11> Section 2.2.53:](#) On Windows 7 and Windows Server 2008 R2 the **wszPhase1AuthSet**, **wszPhase2AuthSet**, and **wszPhase2CryptoSet** sizes are less than 255 characters. On Windows Vista and Windows Server 2008 they are less than 1000 characters.

[<12> Section 2.2.57:](#) On Windows Vista and Windows Server 2008, the only duplicate check performed was for the anonymous method.

[<13> Section 2.2.57:](#) On Windows Vista and Windows Server 2008, the only duplicate check performed was for the anonymous method.

[<14> Section 2.2.75:](#) Windows Vista chooses enforcement state objects to be filter handles provided by the Windows Filtering Platform (for more information, see [\[MSWFPSDK\]](#)).

[<15> Section 3.1.1:](#) Windows Vista selects enforcement state objects to be filter handles provided by the Windows Filtering Platform (for more information, see [\[MSWFPSDK\]](#)).

[<16> Section 3.1.3:](#) Windows Vista selects enforcement state objects to be filter handles provided by the Windows Filtering Platform (for more information, see [\[MSWFPSDK\]](#)).

[<17> Section 3.1.4.6:](#) Path validations were not performed in Windows Vista and Windows Server 2008 at edit time.

[<18> Section 3.1.4.47:](#) Path validations were not performed in Windows Vista and Windows Server 2008 at edit time.

8 Change Tracking

This section identifies changes that were made to the [MS-FASP] protocol document between the January 2011 and February 2011 releases. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements or functionality.
- An extensive rewrite, addition, or deletion of major portions of content.
- The removal of a document from the documentation set.
- Changes made for template compliance.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **Editorial** means that the language and formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical or language changes were introduced. The technical content of the document is identical to the last released version, but minor editorial and formatting changes, as well as updates to the header and footer information, and to the revision summary, may have been made.

Major and minor changes can be described further using the following change types:

- New content added.
- Content updated.
- Content removed.
- New product behavior note added.
- Product behavior note updated.
- Product behavior note removed.
- New protocol syntax added.
- Protocol syntax updated.
- Protocol syntax removed.
- New content added due to protocol revision.
- Content updated due to protocol revision.
- Content removed due to protocol revision.
- New protocol syntax added due to protocol revision.

- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.
- New content added for template compliance.
- Content updated for template compliance.
- Content removed for template compliance.
- Obsolete document removed.

Editorial changes are always classified with the change type **Editorially updated**.

Some important terms used in the change type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.
- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

The changes made to this document are listed in the following table. For more information, please contact protocol@microsoft.com.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
1.2.2 Informative References	60154 Added reference [MSDN-IPV6PROTLEVEL].	N	Content updated.
1.2.2 Informative References	60578 Added reference [MSDN-C++NUMS/OPERS].	N	Content updated.
1.7 Versioning and Capability Negotiation	58989 Clarified that Protocol Versions are used as Binary Versions and Schema Versions. Also clarified the byte order used in this specification.	Y	Content updated.
2.2.2 FW_PROFILE_TYPE	57512 Clarified content to show that the enumeration values can be a single value or a combination of bitmask values, and which combinations are valid.	N	Content updated.
2.2.14 FW_PORT_KEYWORD	59617 Clarified the definition of FW_PORT_KEYWORD_IP_TLS_IN.	N	Content updated.
2.2.14	59618	N	Conte

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
FW_PORT_KEYWORD	Clarified the definition of FW_PORT_KEYWORD_IP_TLS_OUT.		Content updated.
2.2.14 FW_PORT_KEYWORD	59627 Clarified that this enumeration uses bitmask flags to identify the ports.	N	Content updated.
2.2.31 FW_RULE_ORIGIN_TYPE	60578 Clarified the definition of RTL_NUMBER_OF.	N	Content updated.
2.2.34 FW_RULE_FLAGS	60154 Added a reference to [MSDN-IPV6PROTLEVEL] in the product behavior note for FW_RULE_FLAGS_ROUTEABLE_ADDRS_TRAVERSE_DEFER_APP.	N	Product behavior note updated.
2.2.37 FW_PROFILE_CONFIG	59366 Clarified the description of FW_PROFILE_CONFIG_ALLOW_LOCAL_IPSEC_POLICY_MERGE.	N	Content updated.
2.2.37 FW_PROFILE_CONFIG	59367 Clarified the description of FW_PROFILE_CONFIG_ALLOW_LOCAL_POLICY_MERGE.	N	Content updated.
2.2.38 FW_GLOBAL_CONFIG_IPSEC_EXEMPT_VALUES	58704 Clarified that FW_GLOBAL_CONFIG_IPSEC_EXEMPT_DHCP applies to both IPv4 and IPv6 traffic.	N	Content updated.
2.2.38 FW_GLOBAL_CONFIG_IPSEC_EXEMPT_VALUES	58784 Clarified that "FW_GLOBAL_CONFIG_IPSEC_EXEMPT_MAX_V2" is in fact "FW_GLOBAL_CONFIG_IPSEC_EXEMPT_MAX_V2_0".	N	Content updated.
2.2.41 FW_GLOBAL_CONFIG	58951 Clarified that FW_GLOBAL_CONFIG_IPSEC_TUNNEL_REMOTE_MACHINE_AUTHORIZATION_LIST specifies the remote machines.	N	Content updated.
2.2.41 FW_GLOBAL_CONFIG	58952 Clarified that FW_GLOBAL_CONFIG_IPSEC_TUNNEL_REMOTE_USER_	N	Content updated.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
	AUTHORIZATION_LIST specifies the remote users.		ed.
2.2.41 FW_GLOBAL_CONFIG	58961 Revised the definition of FW_GLOBAL_CONFIG_IPSEC_TUNNEL_REMOTE_MACHINE_AUTHORIZATION_LIST to include information about the list of remote machines.	N	Content updated.
2.2.41 FW_GLOBAL_CONFIG	58962 Revised the definition of FW_GLOBAL_CONFIG_IPSEC_TUNNEL_REMOTE_USER_AUTHORIZATION_LIST to include information about the list of remote users.	N	Content updated.
2.2.53 FW_CS_RULE	57149 Clarified the description of PlatformValidityList.	Y	Content updated.
2.2.58 FW_CRYPTO_KEY_EXCHANGE_TYPE	57091 Updated the FW_CRYPTO_KEY_EXCHANGE_DH14 = FW_CRYPTO_KEY_EXCHANGE_DH2048 enumeration.	Y	Content updated.
2.2.78 FW_CONN_HANDLE	58919 Added a reference for "RPC binding handle".	Y	Content updated.
3.1.1 Abstract Data Model	57962 Clarified content to show that the order in which stores are loaded does not matter and that main mode rules are also a type of policy object.	Y	Content updated.
3.1.1 Abstract Data Model	61417 Clarified that FW_STORE_TYPE_NOT_USED_VALUE_3 is not used by the server.	N	Content updated.
3.1.1 Abstract Data Model	61418 Clarified that FW_STORE_TYPE_NOT_USED_VALUE_4 is not used by the server.	N	Content updated.
3.1.3 Initialization	57962 Added an explanation of "store" and clarified that the order of stores does not matter.	Y	Content updated.
3.1.3 Initialization	57963 Explained how [MS-GPFAS] relates to policy stores.	Y	Content updated.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
			ed.
3.1.3 Initialization	58025 Added specific [MS-GPREG] section references concerning timing, reliability, and time to propagate policy changes.	Y	Content updated.
3.1.4.1 RRPC FWOpenPolicyStore (Opnum 0)	58989 Clarified the definition of BinaryVersion.	Y	Content updated.
3.1.4.1 RRPC FWOpenPolicyStore (Opnum 0)	58990 Clarified that dwFlags is reserved for future use.	Y	Content updated.
3.1.4.1 RRPC FWOpenPolicyStore (Opnum 0)	62298 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.2 RRPC FWClosePolicyStore (Opnum 1)	61416 Clarified the definition of this method.	N	Content updated.
3.1.4.2 RRPC FWClosePolicyStore (Opnum 1)	62299 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.3 RRPC FWRestoreDefaults (Opnum 2)	62300 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.4 RRPC FWGetGlobalConfig (Opnum 3)	62301 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.5 RRPC FWSetGlobalConfig (Opnum 4)	62302 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.6 RRPC FWAddFirewallRule (Opnum 5)	62303 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
3.1.4.7 RRPC_FWSetFirewallRule (Opnum 6)	62305 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.8 RRPC_FWDeleteFirewallRule (Opnum 7)	62306 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.9 RRPC_FWDeleteAllFirewallRules (Opnum 8)	62307 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.10 RRPC_FWEnumFirewallRules (Opnum 9)	62308 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.11 RRPC_FWGetConfig (Opnum 10)	62309 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.12 RRPC_FWSetConfig (Opnum 11)	62310 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.13 RRPC_FWAddConnectionSecurityRule (Opnum 12)	62311 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.14 RRPC_FWSetConnectionSecurityRule (Opnum 13)	62312 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.15 RRPC_FWDeleteConnectionSecurityRule (Opnum 14)	62313 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.16 RRPC_FWDeleteAllConnectionSecurityRules (Opnum 15)	62314 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.17 RRPC_FWEnumConnectionSecurityRules (Opnum 16)	62315 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
ityRules (Opnum 16)	appropriate credentials.		updated.
3.1.4.18 RRPC FWAddAuthenticationSet (Opnum 17)	62316 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.19 RRPC FWSetAuthenticationSet (Opnum 18)	62317 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.20 RRPC FWDeleteAuthenticationSet (Opnum 19)	62318 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.21 RRPC FWDeleteAllAuthenticationSets (Opnum 20)	62319 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.22 RRPC FWEnumAuthenticationSets (Opnum 21)	62320 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.23 RRPC FWAddCryptoSet (Opnum 22)	62321 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.24 RRPC FWSetCryptoSet (Opnum 23)	62324 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.25 RRPC FWDeleteCryptoSet (Opnum 24)	62326 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.26 RRPC FWDeleteAllCryptoSets (Opnum 25)	62328 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.27 RRPC FWEnumCryptoSets (Opnum 26)	62330 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
3.1.4.28 RRPC_FWEnumPhase1SAs (Opnum 27)	62331 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.29 RRPC_FWEnumPhase2SAs (Opnum 28)	62332 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.30 RRPC_FWDDeletePhase1SAs (Opnum 29)	62333 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.31 RRPC_FWDDeletePhase2SAs (Opnum 30)	62527 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.32 RRPC_FWEnumProducts (Opnum 31)	62528 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.33 RRPC_FWAddMainModeRule (Opnum 32)	62529 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.34 RRPC_FWSetMainModeRule (Opnum 33)	62530 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.35 RRPC_FWDDeleteMainModeRule (Opnum 34)	62531 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.36 RRPC_FWDDeleteAllMainModeRules (Opnum 35)	62532 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.37 RRPC_FWEnumMainModeRules (Opnum 36)	62533 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.38 RRPC_FWQueryFirewallRules	62534 Clarified how the server validates that the client has	Y	Content

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
(Opnum 37)	appropriate credentials.		updated.
3.1.4.39 RRPC_FWQueryConnectionSecurityRules (Opnum 38)	62535 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.40 RRPC_FWQueryMainModeRules (Opnum 39)	62536 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.41 RRPC_FWQueryAuthenticationSets (Opnum 40)	62537 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.42 RRPC_FWQueryCryptoSets (Opnum 41)	62930 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.43 RRPC_FWEnumNetworks (Opnum 42)	62931 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.44 RRPC_FWEnumAdapters (Opnum 43)	62937 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.45 RRPC_FWGetGlobalConfig2_10 (Opnum 44)	62938 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.46 RRPC_FWGetConfig2_10 (Opnum 45)	62939 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.47 RRPC_FWAddFirewallRule2_10 (Opnum 46)	62940 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.48 RRPC_FWSetFirewallRule2_10 (Opnum 47)	62941 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
3.1.4.49 RRPC_FWEnumFirewallRules2_10 (Opnum 48)	62942 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.50 RRPC_FWAddConnectionSecurityRule2_10 (Opnum 49)	62943 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.51 RRPC_FWSetConnectionSecurityRule2_10 (Opnum 50)	62944 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.52 RRPC_FWEnumConnectionSecurityRules2_10 (Opnum 51)	62945 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.53 RRPC_FWAddAuthenticationSet2_10 (Opnum 52)	62958 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.54 RRPC_FWSetAuthenticationSet2_10 (Opnum 53)	62961 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.55 RRPC_FWEnumAuthenticationSets2_10 (Opnum 54)	62962 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.56 RRPC_FWAddCryptoSet2_10 (Opnum 55)	62964 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.57 RRPC_FWSetCryptoSet2_10 (Opnum 56)	62965 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.
3.1.4.58 RRPC_FWEnumCryptoSets2_10 (Opnum 57)	62967 Clarified how the server validates that the client has appropriate credentials.	Y	Content updated.

9 Index

A

Abstract data model
 [client](#) 165
 [server](#) 94
[Adding firewall rule example](#) 166
[Applicability](#) 12

C

[Capability negotiation](#) 12
[Change tracking](#) 210
Client
 [abstract data model](#) 165
 [initialization](#) 165
 [local events](#) 165
 [message processing](#) 165
 [sequencing rules](#) 165
 [timer events](#) 165
 [timers](#) 165
[Closing policy store example](#) 168

D

Data model - abstract
 [client](#) 165
 [server](#) 94
[Data types](#) 14

E

[Enumerating firewall rules example](#) 168
Examples
 [adding firewall rule example](#) 166
 [closing policy store example](#) 168
 [enumerating firewall rules example](#) 168
 [opening policy store example](#) 166

F

[Fields - vendor-extensible](#) 12
[Full IDL](#) 171
[FW_ADAPTER structure](#) 56
[FW_ADDRESS_KEYWORD enumeration](#) 23
[FW_ADDRESSES structure](#) 24
[FW_AUTH_INFO structure](#) 81
[FW_AUTH_METHOD enumeration](#) 64
[FW_AUTH_SET structure](#) 67
[FW_AUTH_SUITE structure](#) 66
[FW_AUTH_SUITE_FLAGS enumeration](#) 65
[FW_BYTE_BLOB structure](#) 80
[FW_CERT_INFO structure](#) 81
[FW_CONFIG_FLAGS enumeration](#) 55
[FW_COOKIE_PAIR structure](#) 80
[FW_CRYPTO_ENCRYPTION_TYPE enumeration](#) 71
[FW_CRYPTO_HASH_TYPE enumeration](#) 72
[FW_CRYPTO_KEY_EXCHANGE_TYPE enumeration](#) 70

[FW_CRYPTO_PROTOCOL_TYPE enumeration](#) 73
[FW_CRYPTO_SET structure](#) 76
[FW_CS_RULE structure](#) 61
[FW_CS_RULE_ACTION enumeration](#) 59
[FW_CS_RULE_FLAGS enumeration](#) 58
[FW_CS_RULE2_0 structure](#) 60
[FW_DATA_TYPE enumeration](#) 89
[FW_DIRECTION enumeration](#) 22
[FW_ENDPOINTS structure](#) 82
[FW_ENFORCEMENT_STATE enumeration](#) 36
[FW_ENUM_RULES_FLAGS enumeration](#) 41
[FW_GLOBAL_CONFIG enumeration](#) 53
[FW_GLOBAL_CONFIG_IPSEC_EXEMPT_VALUES enumeration](#) 51
[FW_GLOBAL_CONFIG_IPSEC_THROUGH_NAT_VALU ES enumeration](#) 53
[FW_GLOBAL_CONFIG_PRESHARED_KEY_ENCODING_VALUES enumeration](#) 52
[FW_ICMP_TYPE_CODE structure](#) 21
[FW_ICMP_TYPE_CODE_LIST structure](#) 21
[FW_INTERFACE_LUIDS structure](#) 22
[FW_INTERFACE_TYPE enumeration](#) 23
[FW_IP_VERSION enumeration](#) 58
[FW_IPSEC_PHASE enumeration](#) 58
[FW_IPV4_ADDRESS_RANGE structure](#) 18
[FW_IPV4_RANGE_LIST structure](#) 18
[FW_IPV4_SUBNET structure](#) 16
[FW_IPV4_SUBNET_LIST structure](#) 17
[FW_IPV6_ADDRESS_RANGE structure](#) 18
[FW_IPV6_RANGE_LIST structure](#) 19
[FW_IPV6_SUBNET structure](#) 17
[FW_IPV6_SUBNET_LIST structure](#) 17
[FW_MATCH_KEY enumeration](#) 88
[FW_MATCH_TYPE enumeration](#) 90
[FW_MATCH_VALUE structure](#) 89
[FW_MM_RULE structure](#) 86
[FW_NETWORK structure](#) 56
[FW_OBJECT_CTRL_FLAG enumeration](#) 36
[FW_OBJECT_METADATA structure](#) 38
[FW_OS_PLATFORM structure](#) 40
[FW_OS_PLATFORM_LIST structure](#) 40
[FW_OS_PLATFORM_OP enumeration](#) 39
[FW_PHASE1_CRYPTO_FLAGS enumeration](#) 75
[FW_PHASE1_CRYPTO_SUITE structure](#) 73
[FW_PHASE1_KEY_MODULE_TYPE enumeration](#) 80
[FW_PHASE1_SA_DETAILS structure](#) 82
[FW_PHASE2_CRYPTO_PFS enumeration](#) 75
[FW_PHASE2_CRYPTO_SUITE structure](#) 74
[FW_PHASE2_SA_DETAILS structure](#) 84
[FW_PHASE2_TRAFFIC_TYPE enumeration](#) 83
[FW_POLICY_ACCESS_RIGHT enumeration](#) 16
[FW_PORT_KEYWORD enumeration](#) 20
[FW_PORT_RANGE structure](#) 19
[FW_PORT_RANGE_LIST structure](#) 20
[FW_PORTS structure](#) 21
[FW_PRODUCT structure](#) 57
[FW_PROFILE_CONFIG enumeration](#) 49
[FW_PROFILE_TYPE enumeration](#) 15

[FW_QUERY structure](#) 92
[FW_QUERY_CONDITION structure](#) 90
[FW_QUERY_CONDITIONS structure](#) 91
[FW_RULE structure](#) 45
[FW_RULE_ACTION enumeration](#) 42
[FW_RULE_CATEGORY enumeration](#) 56
[FW_RULE_FLAGS enumeration](#) 42
[FW_RULE_ORIGIN_TYPE enumeration](#) 40
[FW_RULE_STATUS enumeration](#) 25
[FW_RULE_STATUS_CLASS enumeration](#) 35
[FW_RULE2_0 structure](#) 44
[FW_STORE_TYPE enumeration](#) 14

G

[Glossary](#) 8

I

[IDL](#) 171
[Implementer - security considerations](#) 170
[Index of security parameters](#) 170
[Informative references](#) 9
Initialization
 [client](#) 165
 [server](#) 95
[Introduction](#) 8

L

Local events
 [client](#) 165
 [server](#) 164

M

Message processing
 [client](#) 165
 [server](#) 96
Messages
 [data types](#) 14
 [transport](#) 14

N

[Normative references](#) 8

O

[Opening policy store example](#) 166
[Overview](#) 10

P

[Parameters - security index](#) 170
[PFW_ADAPTER](#) 56
[PFW_ADDRESSES](#) 24
[PFW_AUTH_INFO](#) 81
[PFW_AUTH_SET](#) 67
[PFW_AUTH_SUITE](#) 66
[PFW_BYTE_BLOB](#) 80
[PFW_CERT_INFO](#) 81

[PFW_COOKIE_PAIR](#) 80
[PFW_CRYPTO_SET](#) 76
[PFW_CS_RULE](#) 61
[PFW_CS_RULE2_0](#) 60
[PFW_ENDPOINTS](#) 82
[PFW_ICMP_TYPE_CODE_LIST](#) 21
[PFW_ICMP_TYPE_CODE_LIST](#) 21
[PFW_INTERFACE_LUIDS](#) 22
[PFW_IPV4_ADDRESS_RANGE](#) 18
[PFW_IPV4_RANGE_LIST](#) 18
[PFW_IPV4_SUBNET](#) 16
[PFW_IPV4_SUBNET_LIST](#) 17
[PFW_IPV6_ADDRESS_RANGE](#) 18
[PFW_IPV6_RANGE_LIST](#) 19
[PFW_IPV6_SUBNET](#) 17
[PFW_IPV6_SUBNET_LIST](#) 17
[PFW_MM_RULE](#) 86
[PFW_NETWORK](#) 56
[PFW_OBJECT_METADATA](#) 38
[PFW_OS_PLATFORM](#) 40
[PFW_OS_PLATFORM_LIST](#) 40
[PFW_PHASE1_CRYPTO_SUITE](#) 73
[PFW_PHASE1_SA_DETAILS](#) 82
[PFW_PHASE2_CRYPTO_SUITE](#) 74
[PFW_PHASE2_SA_DETAILS](#) 84
[PFW_PORT_RANGE](#) 19
[PFW_PORT_RANGE_LIST](#) 20
[PFW_PORTS](#) 21
[PFW_PRODUCT](#) 57
[PFW_QUERY](#) 92
[PFW_QUERY_CONDITION](#) 90
[PFW_QUERY_CONDITIONS](#) 91
[PFW_RULE](#) 45
[PFW_RULE2_0](#) 44
[Preconditions](#) 11
[Prerequisites](#) 11
[Product behavior](#) 205

R

References
 [informative](#) 9
 [normative](#) 8
[Relationship to other protocols](#) 11
[RRPC FWAddAuthenticationSet method](#) 120
[RRPC FWAddAuthenticationSet2_10 method](#) 158
[RRPC FWAddConnectionSecurityRule method](#) 115
[RRPC FWAddConnectionSecurityRule2_10 method](#) 154
[RRPC FWAddCryptoSet method](#) 126
[RRPC FWAddCryptoSet2_10 method](#) 161
[RRPC FWAddFirewallRule method](#) 107
[RRPC FWAddFirewallRule2_10 method](#) 151
[RRPC FWAddMainModeRule method](#) 136
[RRPC FWClosePolicyStore method](#) 103
[RRPC FWDeleteAllAuthenticationSets method](#) 123
[RRPC FWDeleteAllConnectionSecurityRules method](#) 118
[RRPC FWDeleteAllCryptoSets method](#) 129
[RRPC FWDeleteAllFirewallRules method](#) 110
[RRPC FWDeleteAllMainModeRules method](#) 139
[RRPC FWDeleteAuthenticationSet method](#) 122

[RRPC FWDeleteConnectionSecurityRule method](#) 117
[RRPC FWDeleteCryptoSet method](#) 128
[RRPC FWDeleteFirewallRule method](#) 109
[RRPC FWDeleteMainModeRule method](#) 138
[RRPC FWDeletePhase1SAs method](#) 134
[RRPC FWDeletePhase2SAs method](#) 135
[RRPC FWEnumAdapters method](#) 147
[RRPC FWEnumAuthenticationSets method](#) 124
[RRPC FWEnumAuthenticationSets2_10 method](#) 160
[RRPC FWEnumConnectionSecurityRules method](#) 119
[RRPC FWEnumConnectionSecurityRules2_10 method](#) 157
[RRPC FWEnumCryptoSets method](#) 130
[RRPC FWEnumCryptoSets2_10 method](#) 163
[RRPC FWEnumFirewallRules method](#) 111
[RRPC FWEnumFirewallRules2_10 method](#) 153
[RRPC FWEnumMainModeRules method](#) 140
[RRPC FWEnumNetworks method](#) 146
[RRPC FWEnumPhase1SAs method](#) 132
[RRPC FWEnumPhase2SAs method](#) 133
[RRPC FWEnumProducts method](#) 135
[RRPC FWGetConfig method](#) 112
[RRPC FWGetConfig2_10 method](#) 149
[RRPC FWGetGlobalConfig method](#) 104
[RRPC FWGetGlobalConfig2_10 method](#) 148
[RRPC FWOpenPolicyStore method](#) 102
[RRPC FWQueryAuthenticationSets method](#) 144
[RRPC FWQueryConnectionSecurityRules method](#) 142
[RRPC FWQueryCryptoSets method](#) 145
[RRPC FWQueryFirewallRules method](#) 141
[RRPC FWQueryMainModeRules method](#) 143
[RRPC FWRestoreDefaults method](#) 104
[RRPC FWSetAuthenticationSet method](#) 121
[RRPC FWSetAuthenticationSet2_10 method](#) 159
[RRPC FWSetConfig method](#) 113
[RRPC FWSetConnectionSecurityRule method](#) 116
[RRPC FWSetConnectionSecurityRule2_10 method](#) 155
[RRPC FWSetCryptoSet method](#) 127
[RRPC FWSetCryptoSet2_10 method](#) 162
[RRPC FWSetFirewallRule method](#) 108
[RRPC FWSetFirewallRule2_10 method](#) 152
[RRPC FWSetGlobalConfig method](#) 106
[RRPC FWSetMainModeRule method](#) 137

S

Security
[implementer considerations](#) 170
[parameter index](#) 170
Sequencing rules
[client](#) 165
[server](#) 96
Server
[abstract data model](#) 94
[initialization](#) 95
[local events](#) 164
[message processing](#) 96
[sequencing rules](#) 96

[timer events](#) 164
[timers](#) 95
[Standards assignments](#) 13

T

Timer events
[client](#) 165
[server](#) 164
Timers
[client](#) 165
[server](#) 95
[Tracking changes](#) 210
[Transport](#) 14

V

[Vendor-extensible fields](#) 12
[Versioning](#) 12