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

Revision Summary

Date	Revision History	Revision Class	Comments
04/03/2007	0.01		MCPP 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.

Date	Revision History	Revision Class	Comments
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.

Table of Contents

1 Introduction

This document describes the protocol for 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 once this policy is enforced.

1.1 Glossary

The following terms are defined in [MS-GLOS]:

Authentication Level
Authentication Service (AS)
Dynamic Endpoint
Endpoint
Globally Unique Identifier (GUID)
Interface Definition Language (IDL)
Microsoft Interface Definition Language (MIDL)
Network Data Representation (NDR)
Opnum
Remote Procedure Call (RPC)
RPC Protocol Sequence
RPC Transfer Syntax
RPC Transport
Security Association (SA)
Security Provider
Universal Unique Identifier (UUID)
Well-Known Endpoint

The following terms are specific to this document:

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as specified 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-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary", March 2007.

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

[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

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

[MS-TERE] Microsoft Corporation, "Teredo Extensions", July 2007.

[MSDN-OSVERSIONINFOEX] Microsoft Corporation, "OSVERSIONINFOEX Structure", http://msdn2.microsoft.com/en-us/library/ms724833.aspx

[MSWFPSDK] Microsoft Corporation, "Windows Filtering Platform", http://msdn2.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 Protocol 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. Connection security includes authentication, integrity protection, and confidentiality (encryption) of communication.

Because both IPsec and firewall 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 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 received from local users or from network-wide policy 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, given that hosts may have received policies from both sources.

Network-wide policies are normally distributed using Group Policy objects that live on active directories of domains. However, there might be workgroups or networks that don't have a domain infrastructure. Even in non-domain joined environments, the network administrator needs to be able to manage the advanced firewall and IPsec policy of a host remotely.

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 Specification is designed and utilized to address the three needs mentioned above. That is, its purpose is to monitor and manage remote host policies. It can manage all the policies that an administrator could manage locally. It also can monitor the specific policies coming from the different sources or monitor them aggregated, all together, to understand and predict expected behavior. And lastly, it can make temporary modifications on the remote host policy to test online fixes and see if they are effective or not.

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 used to read and manage the different policies available. Hence, the user can make method calls that pass new policy objects to be added to the policy, that delete from the policy, and that 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.

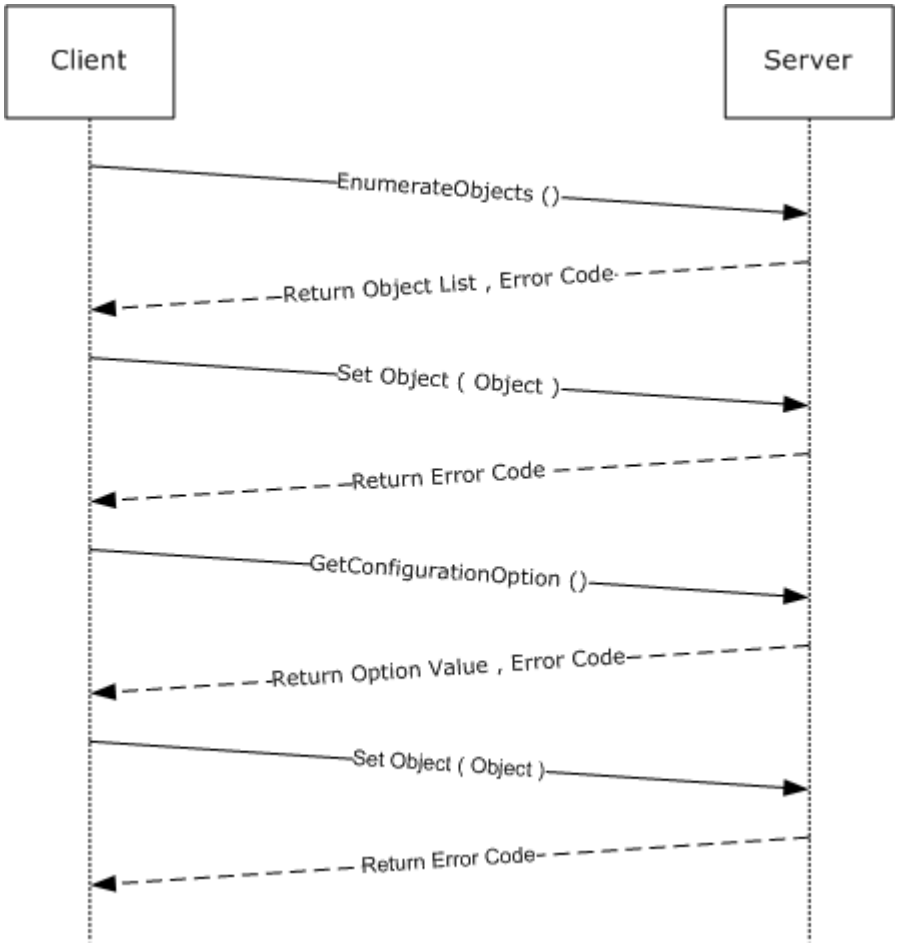

Figure 1: Read and write operations and their message sequences

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 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 (for more information, see [RFC4301]), which describes how IPsec should be enforced and what options the Internet Key Exchange (IKE) (for more information, see [RFC2409]) and AuthIP (as specified in [MS-AIPS]) have 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 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 (running the management tool) will send to the server during this protocol.

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 two policy versions, which can be tied to policies and specific policy objects, as defined in section 2.2. The two versions are 0x0200 and 0x0201.<1>
- Security and Authentication Methods: This protocol supports both Kerberos Protocol Extensions (as specified in [MS-KILE]) and NT LAN Manager (NTLM) Authentication Protocol (as specified in [MS-NLMP]) authentication methods, as defined in 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 2.2.36 contains the maximum policy version 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. Other than this rudimentary capability, there are no protocol version negotiation capabilities or capability negotiation capabilities in this protocol.

1.8 Vendor-Extensible Fields

This protocol uses Win32 error codes. These values are taken from the Windows error number space 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 are free to choose their own values for this field, as long as the C bit (0x20000000) is set, indicating it is a customer code.

Currently, vendors are not expected to extend this protocol. Because of this, the protocol does not consider provisions for extensions by non-Microsoft parties. This might or might not be added in the future.

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

The following sections specify how Firewall and Advanced Security Protocol messages are transported and common Firewall and Advanced Security Protocol data types.

2.1 Transport

This protocol uses the RPC over TCP (ncacn_ip_tcp) remote procedure call protocol sequence. It also uses RPC dynamic endpoints, as specified in [C706] chapters 6, 7, 8, 9, 10, 11, 12, 13, and 14.

This RPC protocol uses 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 this 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

```
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: An invalid value that 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 containing the network-wide 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, hence it is 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, hence it is read and 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 read-only store. This symbolic constant has a value of 7.

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

2.2.2 FW_PROFILE_TYPE

This data type defines enumerations used to identify profile types. The enumeration values are bitmasks and MAY be combined on specific instances. A profile is a set of networks to which a firewall policy might be applicable.

```
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: An invalid value, 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 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 are in some physical private place such as a home or a work 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 there is no trust on the peers in the network or on the network administrator.

FW_PROFILE_TYPE_ALL: This value represents all of the above types of networks and any future types.

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

FW_PROFILE_TYPE_NONE: This value represents no profile. This is an invalid value. 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 Windows Firewall and Advanced Security Protocol. The values are not bitmasks, and SHOULD never 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: An invalid value, 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 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: An invalid value. It is defined for simplicity in writing IDL definitions and code. This and greater values MUST NOT be used. This symbolic constant has a value of 3.

2.2.4 FW_IPV4_SUBNET

This structure defines IPv4 subnets. This structure 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, and 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 **dwNumEntries** contiguous **FW_IPV4_SUBNET** elements.

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 MUST NOT contain either an unspecified address or 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: This field is a pointer to an array of **dwNumEntries** contiguous **FW_IPV6_SUBNET** elements.

2.2.8 FW_IPV4_ADDRESS_RANGE

```
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 in included in the range.

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

2.2.9 FW_IPV4_RANGE_LIST

```
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 **dwNumEntries** contiguous **FW_IPV4_ADDRESS_RANGE** elements.

2.2.10 FW_IPV6_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;
```

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 **Begin** less than or equal to **End**. 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

The **FW_IPV6_RANGE_LIST** structure.

```
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 **dwNumEntries** contiguous **FW_IPV6_ADDRESS_RANGE** elements.

2.2.12 FW_PORT_RANGE

```
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 **wBegin** less than or equal to **wEnd**. In this protocol, **wBegin** is equal to **wEnd**.

2.2.13 FW_PORT_RANGE_LIST

The **FW_PORT_RANGE_LIST** structure.

```
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 **dwNumEntries** contiguous **FW_PORT_RANGE** elements.

2.2.14 FW_PORT_KEYWORD

```
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_MAX = 0x08
} FW_PORT_KEYWORD;
```

FW_PORT_KEYWORD_NONE: Specifies that no port keywords are used.

FW_PORT_KEYWORD_DYNAMIC_RPC_PORTS: Represents ports 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 dynamic RPC ports, see [MS-RPCE].

FW_PORT_KEYWORD_RPC_EP: Represents ports used by RPC **endpoint** mapper only when an RPC interface is registered for the corresponding **RPC transport**. This keyword currently can only represent 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 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 [MS-TERE]) and Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs) (for more information, see [RFC4380]).

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

2.2.15 FW_PORTS

```
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 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 with a specific use.

2.2.17 FW_ICMP_TYPE_CODE_LIST

```
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 port ranges that the structure contains.

pEntries: A pointer to an array of **dwNumEntries** contiguous **FW_ICMP_TYPE_CODE** elements.

2.2.18 FW_INTERFACE_LUIDS

The **FW_INTERFACE_LUIDS** structure.

```
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 port ranges that the structure contains.

pLUIDs: A pointer to an array of **dwNumEntries** contiguous **GUID** elements. The **GUID** data type is specified in [MS-DTYP].

2.2.19 FW_DIRECTION

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

2.2.20 FW_INTERFACE_TYPE

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

2.2.21 FW_ADDRESS_KEYWORD

```
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 currently lie within the computer's local subnet.

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

2.2.22 FW_ADDRESSES

```
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

```
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 CRYPTO = 0x0008000A,
  FW RULE STATUS PARSING ERROR PHASE2 AUTH = 0x0008000B,
  FW RULE STATUS PARSING ERROR RESOLVE APP = 0x0008000C,
  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_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_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 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 PHASE1 AUTH SET ID = 0x00100500,
  FW RULE STATUS SEMANTIC ERROR PHASE2 CRYPTO SET ID = 0x00100510,
  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 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_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 MACHINE USER AUTH = 0x00101070,
  FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPTO_SET_ID = 0x00105000,
  FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPTO_FLAGS = 0x00105001,
  FW RULE STATUS SEMANTIC ERROR PHASE1 CRYPTO TIMEOUT MINUTES = 0x00105002,
  FW RULE STATUS SEMANTIC ERROR PHASE1 CRYPTO TIMEOUT SESSIONS = 0x00105003,
  FW RULE STATUS SEMANTIC ERROR PHASE1 CRYPTO KEY EXCHANGE = 0x00105004,
  FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPTO_ENCRYPTION = 0x00105005,
  FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPTO_HASH = 0x00105006,
  FW RULE STATUS SEMANTIC ERROR PHASE1 CRYPTO ENCRYPTION VER = 0x00105007,
  FW RULE STATUS SEMANTIC ERROR PHASE1 CRYPTO HASH VER = 0x00105008,
  FW RULE STATUS SEMANTIC ERROR PHASE2 CRYPTO PFS = 0x00105020,
  FW RULE STATUS SEMANTIC ERROR PHASE2 CRYPTO PROTOCOL = 0x00105021,
  FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPTO_ENCRYPTION = 0x00105022,
  FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPTO_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_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 = 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 only be present if the policy was written (such as the Group Policy) by future firewall and advanced security components that support a higher schema version. Hence, this error will only occur 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 can't 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 will be understandable to lower major version components.

FW_RULE_STATUS_PARSING_ERROR_NAME: The name contains invalid characters or length.

FW_RULE_STATUS_PARSING_ERROR_DESC: The description contains invalid characters or length.

FW_RULE_STATUS_PARSING_ERROR_APP: The application contains invalid characters or length.

FW_RULE_STATUS_PARSING_ERROR_SVC: The service contains invalid characters or length.

FW_RULE_STATUS_PARSING_ERROR_RMA: The remote machine authentication contains invalid characters or length.

FW_RULE_STATUS_PARSING_ERROR_RUA: The remote user authentication contains invalid characters or length.

FW_RULE_STATUS_PARSING_ERROR_EMBD: The embedded context contains invalid characters or length.

FW_RULE_STATUS_PARSING_ERROR_RULE_ID: The rule ID contains invalid characters or length.

FW_RULE_STATUS_PARSING_ERROR_PHASE1_AUTH: The Phase1 authentication set ID contains invalid characters or length.

FW_RULE_STATUS_PARSING_ERROR_PHASE2_CRYPTO: The Phase2 crypto set ID contains invalid characters or length.

FW_RULE_STATUS_PARSING_ERROR_PHASE2_AUTH: The Phase2 authentication set ID contains invalid characters or length.

FW_RULE_STATUS_PARSING_ERROR_RESOLVE_APP: The application name could not be resolved.

FW_RULE_STATUS_PARSING_ERROR: The rule failed to be parsed correctly.

FW_RULE_STATUS_SEMANTIC_ERROR_RULE_ID: Semantic error: Rule ID not specified.

FW_RULE_STATUS_SEMANTIC_ERROR_PORTS: Semantic error: Mismatch in number of ports and ports buffer.

FW_RULE_STATUS_SEMANTIC_ERROR_PORT_KEYW: Semantic error: Invalid port keyword.

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 number of v4 subnets and subnets buffer.

FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_V6_SUBNETS: Semantic error: Mismatch in number of v6 subnets and subnets buffer.

FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_V4_RANGES: Semantic error: Mismatch in number of v4 ranges and ranges buffer.

FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_V6_RANGES: Semantic error: Mismatch in number of v6 ranges and ranges buffer.

FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_RANGE: Semantic error: End < Begin.

FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_MASK: Semantic error: Invalid mask specified on a v4 subnet.

FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_PREFIX: Semantic error: Invalid prefix specified on a v6 subnet.

FW_RULE_STATUS_SEMANTIC_ERROR_ADDR_KEYW: Semantic error: An invalid keyword is specified.

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 in conjunction with either an interface or 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: Endpoint "any" cannot be specified for a tunnel mode rule.

FW_RULE_STATUS_SEMANTIC_ERROR_PROFILE: Semantic error: Profile type is invalid.

FW_RULE_STATUS_SEMANTIC_ERROR_ICMP: Semantic error: Mismatch in number of ICMP and ICMP buffer.

FW_RULE_STATUS_SEMANTIC_ERROR_ICMP_CODE: Semantic error: Invalid ICMP code specified.

FW_RULE_STATUS_SEMANTIC_ERROR_IF_ID: Semantic error: Mismatch in number of interfaces and interfaces buffer.

FW_RULE_STATUS_SEMANTIC_ERROR_IF_TYPE: Semantic error: Invalid interface type specified.

FW_RULE_STATUS_SEMANTIC_ERROR_ACTION: Semantic error: Invalid action specified.

FW_RULE_STATUS_SEMANTIC_ERROR_ALLOW_BYPASS: Semantic error: Allow-Bypass action specified, but the rule does not meet allow-bypass criteria (inbound, authenticate/encrypt flags set, remote machine authentication list specified).

FW_RULE_STATUS_SEMANTIC_ERROR_DO_NOT_SECURE: Semantic error: Do_not_secure action specified along with authentication and/or crypto sets.

FW_RULE_STATUS_SEMANTIC_ERROR_ACTION_BLOCK_IS_ENCRYPTED_SECURE: Semantic error: Block action was specified in conjunction with require security or require encryption.

FW_RULE_STATUS_SEMANTIC_ERROR_DIR: Semantic error: Invalid direction specified.

FW_RULE_STATUS_SEMANTIC_ERROR_PROT: Semantic error: Invalid protocol specified.

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

FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS: Semantic error: Invalid flags specified.

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

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

FW_RULE_STATUS_SEMANTIC_ERROR_FLAGS_AUTO_DYN_RPC: Semantic error: Autogenerate flag is set along with dynamic RPC flag.

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

FW_RULE_STATUS_SEMANTIC_ERROR_REMOTE_AUTH_LIST: Semantic error: Authorized remote machine or user list specified, but authenticate/encryption flags not set.

FW_RULE_STATUS_SEMANTIC_ERROR_REMOTE_USER_LIST: Semantic error: Authorized remote user list specified on outbound direction.

FW_RULE_STATUS_SEMANTIC_ERROR_PLATFORM: Semantic error: Number of valid OS platforms and the list of valid OS platforms don't match.

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

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPTO_SET_ID: Semantic error: Phase2 crypto set ID not specified.

FW_RULE_STATUS_SEMANTIC_ERROR_SET_ID: Semantic error: Set ID not specified.

FW_RULE_STATUS_SEMANTIC_ERROR_IPSEC_PHASE: Semantic error: Invalid phase specified.

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

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_AUTH_METHOD: Semantic error: Invalid Phase1 authentication method.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_AUTH_METHOD: Semantic error: Invalid Phase2 authentication method.

FW_RULE_STATUS_SEMANTIC_ERROR_AUTH_METHOD_ANONYMOUS: Semantic error: Anonymous authentication specified as sole authentication proposal (suite).

FW_RULE_STATUS_SEMANTIC_ERROR_AUTH_SUITE_FLAGS: Semantic error: Invalid authentication suite flags specified.

FW_RULE_STATUS_SEMANTIC_ERROR_HEALTH_CERT: Semantic error: Machine cert must be health cert for Phase2 authentication.

FW_RULE_STATUS_SEMANTIC_ERROR_AUTH_SIGNCERT_VER: Semantic error: Suite specifies signing not compliant with its schema version.

FW_RULE_STATUS_SEMANTIC_ERROR_MACHINE_SHKEY: Semantic error: Missing or invalid machine shared key.

FW_RULE_STATUS_SEMANTIC_ERROR_CA_NAME: Semantic error: Missing or invalid CA name.

FW_RULE_STATUS_SEMANTIC_ERROR_MIXED_CERTS: Semantic error: Health certs cannot be specified together with regular certs.

FW_RULE_STATUS_SEMANTIC_ERROR_NON_CONTIGUOUS_CERTS: Semantic error: Certs with a specific signing algorithm are not contiguous.

FW_RULE_STATUS_SEMANTIC_ERROR_MACHINE_USER_AUTH: Semantic error: Both machine and user authentication specified.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPTO_SET_ID: Semantic error: Phase1 crypto set ID is not the default.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPTO_FLAGS: Semantic error: Invalid Phase1 crypto set flags.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPTO_TIMEOUT_MINUTES: Semantic error: Invalid Phase1 crypto set time-out minutes.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPTO_TIMEOUT_SESSIONS: Semantic error: Invalid Phase1 crypto set time-out sessions.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPTO_KEY_EXCHANGE: Semantic error: Invalid Phase1 crypto set key exchange.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPTO_ENCRYPTION: Semantic error: Invalid Phase1 crypto set encryption.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPTO_HASH: Semantic error: Invalid Phase1 crypto set hash.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPTO_ENCRYPTION_VER: Semantic error: Not schema version compliant Phase1 Crypto set encryption.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPTO_HASH_VER: Semantic error: Not schema version compliant Phase1 crypto set hash.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPTO_PFS: Semantic error: Invalid Phase2 crypto set PFS.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPTO_PROTOCOL: Semantic error: Invalid Phase2 crypto set protocol.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPTO_ENCRYPTION: Semantic error: Invalid Phase2 crypto set encryption.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPTO_HASH: Semantic error: Invalid Phase2 crypto set hash.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPTO_TIMEOUT_MINUTES: Semantic error: Invalid Phase2 crypto set time-out minutes.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPTO_TIMEOUT_KBYTES: Semantic error: Invalid Phase2 crypto set time-out kilobytes.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPTO_ENCRYPTION_VER: Semantic error: Not schema version compliant Phase2 crypto set encryption.

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPTO_HASH_VER : Not schema version compliant Phase2 crypto set hash.

FW_RULE_STATUS_SEMANTIC_ERROR_CRYPTO_ENCR_HASH: Semantic error: Encryption and hash both not specified.

FW_RULE_STATUS_SEMANTIC_ERROR_CRYPTO_ENCR_HASH_COMPAT : Semantic error: Encryption and hash using incompatible algorithms.

FW_RULE_STATUS_SEMANTIC_ERROR_SCHEMA_VERSION: Semantic error: Schema version specified is below the supported versions.

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

FW_RULE_STATUS_RUNTIME_ERROR_PHASE1_AUTH_NOT_FOUND: Phase1 authentication set was not found.

FW_RULE_STATUS_RUNTIME_ERROR_PHASE2_AUTH_NOT_FOUND: Phase2 authentication set was not found.

FW_RULE_STATUS_RUNTIME_ERROR_PHASE2_CRYPTO_NOT_FOUND: Phase2 crypto set was not found.

FW_RULE_STATUS_RUNTIME_ERROR_AUTH_MCHN_SHKEY_MISMATCH: There is a runtime error when the rule is in conjunction with other policy objects.

FW_RULE_STATUS_RUNTIME_ERROR: There is a runtime error when the object is considered in conjunction 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 (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_OS_PLATFORM

This structure identifies an operating system platform.<2>

```
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: Identifies the platform type.

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.

2.2.26 FW_OS_PLATFORM_LIST

```
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.

2.2.27 FW_RULE_ORIGIN_TYPE

```
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, 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 2.2.46 and 2.2.55, respectively.<3> This symbolic constant has a value of 5.

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

2.2.28 FW_ENUM_RULES_FLAGS

```
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_MAX = 0x0040
} FW_ENUM_RULES_FLAGS;
```

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

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 descriptions 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 GPO name for the GP_RSOP rules.

FW_ENUM_RULES_FLAG_EFFECTIVE: Enumerates rules only if an attempt was made to push them to BFE (dynamic store only).

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

2.2.29 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 only applicable 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 is invalid. It is defined for simplicity in writing IDL definitions and code. This and greater values MUST NOT be used. This symbolic constant 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).

2.2.30 FW_RULE_FLAGS

```
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_MAX = 0x0020
} FW_RULE_FLAGS;
```

FW_RULE_FLAGS_NONE: This value means that no flags are set, hence the rule is disabled.

FW_RULE_FLAGS_ACTIVE: The rule is enabled if this flag is set.

FW_RULE_FLAGS_AUTHENTICATE: This flag MUST only be set on rules with 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: Same as the FW_RULE_FLAGS_AUTHENTICATE flag, but traffic also MUST be encrypted.

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

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

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

2.2.31 FW_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;
} 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 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 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 the rule applies to all interfaces; that is, the condition is not applied.

dwLocalInterfaceTypes: Bitmask of **FW_INTERFACE_TYPE**. It is a condition that restricts the interface types used by the traffic that the rule matches. 0x00000000 means the condition matches all interface types.

wszLocalApplication: A pointer to a Unicode string. It's 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 the rule applies to all processes in the host.

wszLocalService: A pointer to a Unicode string. It's 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 the rule applies to all services in the system. A null in this field means 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.

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.

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 by groups.

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

Status: Status code of the rule as specified by **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: 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 representing the name of the originating GPO. It MUST be set if rule origin is GP; otherwise, it MUST be null.

Reserved: Not used. The value is ignored, and reserved for future use.

The following are semantic checks that firewall rules MUST pass:

- **wSchemaVersion** MUST NOT be less than 0x000100.
- **wszRuleId** 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 or equal to 1,000 characters.
- **wszName** field string MUST be at least 1 character long, MUST NOT be greater or equal to 10,000 characters, MUST NOT be NULL, and MUST contain the pipe (|) character.
- **wszName** MUST NOT be equal (case insensitive) to the string "ALL".
- If the **wszDescription** field string is not NULL, it MUST be at least 1 character long, MUST NOT be greater 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 long, MUST NOT be greater or equal to MAX_PATH characters, and MUST NOT contain the following characters: /,*,?,",<,>,|.
- If the **wszLocalService** field string is not NULL, it MUST be at least 1 character long, MUST NOT be greater or equal to MAX_PATH characters, and MUST NOT contain the following characters: /,\,|.
- If the **wszEmbeddedContext** field string is not NULL, it MUST be at least 1 character long, MUST NOT be greater or equal to 10,000 characters, and MUST NOT contain the pipe (|) character.
- **Direction** field MUST NOT contain invalid **FW_DIRECTION** values.
- **dwProfiles** field MUST NOT contain invalid values and, if it is not equal to the ALL profile type, it MUST NOT contain unknown profiles.
- **wIpProtocol** field MUST NOT be greater than 256.
- If the **LocalPort wPortKeywords** field is FW_PORT_KEYWORD_DYNAMIC_RPC_PORTS or FW_PORT_KEYWORD_RPC_EP, the **wIpProtocol** MUST be 6, and **Direction** MUST be FW_DIRECTION_IN.
- If the **LocalPort wPortKeywords** field is FW_PORT_KEYWORD_TEREDO_PORT, the **wIpProtocol** MUST be 17, and **Direction** MUST be FW_DIRECTION_IN.

- The **LocalPort wPortKeywords** MUST be 0 if the **Direction** is FW_DIRECTION_OUT.
- If the **wIpProtocol** field is 6 or 17, the **RemotePort wPortKeywords** field MUST be 0.
- If the **wIpProtocol** field is not 1, 6, 17, nor 58, the **LocalPort**, **RemotePort**, **V4TypeCodeList**, and **V6TypeCodeList** must be empty.
- **LocalAddresses**, **dwV4AddressKeywords**, and **dwV6AddressKeywords** 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 the FW_RULE_FLAGS_AUTHENTICATE or FW_RULE_FLAGS_AUTHENTICATE_WITH_ENCRYPTION, **wFlags** MUST NOT contain FW_RULE_FLAGS_LOOSE_SOURCE_MAPPED.
- **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 long, 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, MUST have a not 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, it MUST be at least 1 character long, 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, MUST have a not 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 **wszRemoteMachineAuthorizationList** 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.

2.2.32 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 these options' values 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; 0x000000001 is on. The merge law for this option is to let the value GP RSOP store win if it is configured; otherwise, use the local store value. This symbolic constant has a value of 1.

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

FW_PROFILE_CONFIG_SHIELDED: This value is a **DWORD**; it is an on/off switch. If on, it blocks all incoming traffic regardless of what the rest of the policy specifies (except if 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**; it is an on/off switch value. If it is on, unicast responses to multicast broadcast traffic is blocked. The merge law for this option is to let the value GP RSOP store win if it is configured; otherwise, use the local store value. This symbolic constant has a value of 4.

FW_PROFILE_CONFIG_LOG_DROPPED_PACKETS: This value is a **DWORD**; it is an on/off switch value. 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**; it is an on/off switch value. 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**; it is 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 GP RSOP store win if it is configured; otherwise, use the local store value. 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 where the firewall logs dropped packets and successful connections. The merge law for this option is to let the value GP RSOP store win if it is configured; otherwise, use the local store value. This symbolic constant has a value of 9.

FW_PROFILE_CONFIG_DISABLE_INBOUND_NOTIFICATIONS: This value is a **DWORD**; it is an on/off switch. If this value is off, whenever an unknown application listens on a port, the firewall pops up a notification to the user who ran the application, prompting for an action. The merge law for this option is to let the value GP RSOP store win if it is configured; otherwise, use the local store value. This symbolic constant has a value of 10.

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

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

FW_PROFILE_CONFIG_ALLOW_LOCAL_POLICY_MERGE: This value is a **DWORD**; it is an on/off switch. If this value is off, firewall rules (current and legacy ones) from the local store are ignored and not enforced. The merge law for this option is to let the value GP RSOP store win if it is configured; otherwise, use the local store value. 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. The merge law for this option is to let the value GP RSOP store win if it is configured; otherwise, use the local store value. This symbolic constant has a value of 14.

FW_PROFILE_CONFIG_DISABLED_INTERFACES: This value is an **FW_INTERFACE_LUIDS** structure representing 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 GP RSOP store win if it is configured; otherwise, use the local store value. 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. 0x00000000 represents the allow action; 0x00000001 represents a block action. The merge law for this option is to let the value GP RSOP store win if it is configured; otherwise, use the local store value. 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. 0x00000000 represents the allow action; 0x00000001 represents a block action. The merge law for this option is to let the value GP RSOP store win if it is configured; otherwise, use the local store value. This symbolic constant has a value of 17.

FW_PROFILE_CONFIG_MAX: This is an invalid value. It is defined for simplicity in writing IDL definitions and code. Values greater than this value are also invalid. This symbolic constant has a value of 18.

2.2.33 FW_GLOBAL_CONFIG_IPSEC_EXEMPT_VALUES

```
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_MAX = 0x0008,
  FW_GLOBAL_CONFIG_IPSEC_EXEMPT_MAX_V2 = 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_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: This symbolic constant is defined for simplicity in writing IDL definitions and describing semantic checks against policy schema versions of 0x0200.

2.2.34 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.

2.2.35 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.

2.2.36 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 these options' values 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_MAX
} FW_GLOBAL_CONFIG;
```

FW_GLOBAL_CONFIG_INVALID: This value MUST NOT 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**: it contains the maximum policy version that the server host is able to understand. 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 given 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 by the server firewall host. This value is only available in the dynamic store, so it is not merged and, hence, 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 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 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 not seeing network traffic for this specified period of time. The value is a **DWORD**, and MUST be a value within the range of 300 to 3,600 inclusive. The merge law for this option is to let the value GP 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 used. The value is a **DWORD**, and MUST be a valid value from **FW_GLOBAL_CONFIG_PRESHARED_KEY_ENCODING_VALUES** enumeration. The merge law for this option is to let the value GP 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 defined in **FW_GLOBAL_CONFIG_IPSEC_EXEMPT_VALUES**; hence, the maximum value MUST always be FW_GLOBAL_CONFIG_IPSEC_EXEMPT_MAX-1 for servers supporting schema version of 0x0201, and to FW_GLOBAL_CONFIG_IPSEC_EXEMPT_MAX_V2_0-1_for servers supporting a schema version of 0x0200. If the maximum value is exceeded when calling the method **RRPC_FWSetGlobalConfig (Opnum 4)**, then the method returns ERROR_INVALID_PARAMETER. This error code is return if no other preceding error is

discovered. The merge law for this option is to let the value GP 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 lists 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 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 GP 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 configures when 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 GP 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, hence, 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 supported by the server. Where different versions of the data types exist, the data type definition in this section specifies the characteristics relevant to this version. This value is not merged. In addition, this value is always a fixed value for a given firewall and advanced security component's software build. This symbolic constant has a value of 11. This value identifies a policy configuration option supported only on servers with schema version of 0x0201,

FW_GLOBAL_CONFIG_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 12.

2.2.37 FW_CONFIG_FLAGS

```
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 specified, if the **RRPC_FWGetConfig (Opnum 10)** method or the **RRPC_FWGetGlobalConfig (Opnum 3)** method fails to find the configuration value in the policy store, the call will succeed and return the default value used by the firewall service. If 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 decision, and is outside the scope of this protocol specification.<4>

2.2.38 FW_IP_VERSION

```
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 the internet protocol version 4. This symbolic constant has a value of 1.

FW_IP_VERSION_V6: This value represents the internet protocol version 6. 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.

2.2.39 FW_IPSEC_PHASE

```
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 IPsec first phase of negotiations, also called main mode. This symbolic constant has a value of 1.

FW_IPSEC_PHASE_2: This value represents IPsec second phase of negotiations, also called quick mode or extended mode. A phase 1 authentication is the second authentication referred to as extended mode in AuthIP, as specified in [MS-AIPS]. 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.

2.2.40 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_MAX = 0x02
```

```
} FW_CS_RULE_FLAGS;
```

FW_CS_RULE_FLAGS_NONE: No flags are specified.

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

FW_CS_RULE_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.

2.2.41 FW_CS_RULE_ACTION

```
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. 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.

2.2.42 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(1,1001), 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_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 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 this condition is not applied (any match).

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 the rule applies to all interfaces; that is, the condition is not applied.

dwLocalInterfaceTypes: 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 will 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 will 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 will 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 will 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 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.

wszPhase1AuthSet: A Unicode string that represents the set identifier of a Phase1 authentication sets policy objects.

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

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

Action: The connection security action the rule will take for the traffic matches. This field must contain a valid value from the **FW_CS_RULE_ACTION** enumeration.

wFlags: Bit 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 group by groups.

PlatformValidityList: A condition that specifies the platforms sending or receiving 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 representing the name of the originating GPO. It MUST be set if rule origin is GP; otherwise, it MUST be null.

Status: Status code of the rule, as specified by **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.

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

- **wSchemaVersion** MUST NOT be less than 0x000200.
- **wszRuleId** 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 1,000 characters.
- **wszName** field string MUST be at least 1 character long, MUST NOT be greater than or equal to 10,000 characters, MUST NOT be NULL, and MUST contain the pipe (|) character.
- **wszName** MUST NOT be equal (case insensitive) to the string "ALL".
- 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.
- **dwProfiles** field MUST NOT contain invalid values and, if it is not equal to the ALL profile type, it MUST NOT contain unknown profiles.
- **wIpProtocol** field MUST NOT be greater than 256.
- If the **wIpProtocol** is 6 or 17, **theEndpoint1Ports** and **wPortKeywords** fields MUST be 0.
- If the **wIpProtocol** is 6 or 17, the **Endpoint2Ports** and **wPortKeywords** fields MUST be 0.
- If the **wIpProtocol** is not 6 nor 17, the **Endpoint1Ports** and **Endpoint2Ports** fields must be empty.
- If **Endpoint1** field is not empty, **LocalInterfaceIds** MUST be empty and **dwLocalInterfaceTypes** MUST be 0. The opposite also applies.
- **Endpoint1** and **Endpoint2** addresses keywords MUST contain valid address keywords.

- **Endpoint1** and **Endpoint2** addresses MUST NOT contain multicast v4 and v6 addresses.
- **dwLocalInterfaceTypes** MUST NOT be greater than or equal to FW_INTERFACE_TYPE_MAX.
- **Action** MUST be a valid action from the **FW_CS_RULE_ACTION** enumeration.
- **wFlags** MUST NOT be greater than FW_CS_RULE_FLAGS_MAX.
- If the **Action** field is FW_CS_RULE_ACTION_DO_NOT_SECURE, the **wszPhase1AuthSet**, the **wszPhase2AuthSet**, and the **wszPhase2CryptoSet** MUST all be NULL; otherwise, **wszPhase1AuthSet** and **wszPhase2CryptoSet** MUST NOT be NULL.
- **wszPhase1AuthSet**, **wszPhase2AuthSet**, and **wszPhase2CryptoSet** MUST all be at least 1 character long, MUST NOT be greater than or equal to 1,000 characters, and MUST NOT contain the pipe (|) character.
- If the rule is a tunnel rule, **Endpoint1** and **Endpoint2** addresses MUST NOT be empty, **Action** 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 endpoints.
- Tunnel endpoint addresses must not be loopback addresses.

2.2.43 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 the machines using the Kerberos Protocol Extensions (as specified in [MS-KILE]) authentication protocol. This method MUST only be used 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 only be used on first authentications. This symbolic constant has a value of 3.

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

FW_AUTH_METHOD_MACHINE_CERT: This method authenticates the identity of the machines 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 using the Kerberos Protocol Extensions (as specified in [MS-KILE]) authentication protocol. This method MUST only be used on second authentications. This symbolic constant has a value of 6.

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

FW_AUTH_METHOD_USER_NTLM: This method authenticates user identities using the NT LAN Manager (NTLM) Authentication Protocol, as specified in [MS-NLMP]. This method MUST only be used 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. It is defined for simplicity in writing IDL definitions and code. This symbolic constant has a value of 9.

2.2.44 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_MAX = 0x0020
} FW_AUTH_SUITE_FLAGS;
```

FW_AUTH_SUITE_FLAGS_NONE: This value means that no flags are specified.

FW_AUTH_SUITE_FLAGS_CERT_EXCLUDE_CA_NAME: If this flag is set, certificate authority names are excluded. This flag MUST only be set 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 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 using the 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 the curves with a 384-bit prime moduli.

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.

2.2.45 FW_AUTH_SUITE

This structre describes an IPsec authentication suite. An authentication suite is a proposal of a set of algorithm and parameters that specifies the authentication method to be used, some modifiers, and some parameters for the same 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 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 used to authenticate when using pre-shared key methods.

If the method is machine certificate or user certificate, the **wszCAName** string MUST NOT be NULL, MUST at least be 1 character long, MUST NOT be greater than 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 10,000 characters, and MUST NOT contain the pipe (|) character.

2.2.46 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 the structure contains.

pSuites: A pointer to an array of **dwNumSuites** contiguous **FW_AUTH_SUITE** elements.

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 representing the name of the originating GPO. It MUST be set if the origin is GP; otherwise, it MUST be null.

Status: Status code of the set, as specified by **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: Reserved value, and not currently used. It MUST be set to 0.

The following are semantic checks that authentication sets MUST pass:

- **wSchemaVersion** MUST NOT be less than 0x000200.
- **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 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** 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 with the 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, then the **wFlags** field MUST NOT contain neither the FW_AUTH_SUITE_FLAGS_CERT_SIGNING_ECDSA256, nor the FW_AUTH_SUITE_FLAGS_CERT_SIGNING_ECDSA384 flags.
- The **wFlags** field cannot contain both the FW_AUTH_SUITE_FLAGS_CERT_SIGNING_ECDSA256 and the FW_AUTH_SUITE_FLAGS_CERT_SIGNING_ECDSA384 flags.
- All suites with a FW_AUTH_METHOD_MACHINE_CERT method and a **wFlags** field with FW_AUTH_SUITE_FLAGS_CERT_SIGNING_ECDSA256 flag set MUST be contiguous. The same applies for those suites with the FW_AUTH_SUITE_FLAGS_CERT_SIGNING_ECDSA384 flag set, and those suites with neither flag set (which defaults to RSA signing).
- All such contiguous suites with a specific signing flag (either none, ECDSA256 or ECDSA384) need tp agree to either include, or not include, the FW_AUTH_SUITE_FLAG_HEALTH_CERT flag.
- The set MUST NOT have more than one suite with the anonymous method.
- The set MUST NOT have a suite with an NTLM Authentication Protocol (as specified in [MS-NLMP]) method and a suite SHKey method.
- If the set has a machine cert suite that has a **wFlag** containing 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 with the anonymous method as the only suite.
  - Suites in the set MUST NOT contain the FW_AUTH_SUITE_FLAGS_CERT_EXCLUDE_CA_NAME.
  - Suites with user certificate methods MUST NOT contain the FW_AUTH_SUITE_FLAGS_HEALTH_CERT flag while suites with machine certificate methods MUST contain it.
  - Authentication suites with the 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, then the **wFlags** field MUST NOT contain neither the FW_AUTH_SUITE_FLAGS_CERT_SIGNING_ECDSA256, nor the FW_AUTH_SUITE_FLAGS_CERT_SIGNING_ECDSA384 flags.

- The **wFlags** field cannot contain both the FW_AUTH_SUITE_FLAGS_CERT_SIGNING_ECDSA256 and the FW_AUTH_SUITE_FLAGS_CERT_SIGNING_ECDSA384 flags.
- All suites with a FW_AUTH_METHOD_MACHINE_CERT method and a **wFlags** field with FW_AUTH_SUITE_FLAGS_CERT_SIGNING_ECDSA256 flag set MUST be contiguous. The same applies for those suites with the FW_AUTH_SUITE_FLAGS_CERT_SIGNING_ECDSA384 flag set, and those suites with neither flag set (which defaults to RSA signing).
- The set MUST NOT have more than one suite with the anonymous method.
- A set containing a suite with the machine certificate method MUST NOT contain suites with the user certificate method.
- A set containing a suite with the machine certificate method MUST only contain more suites with machine certificate or anonymous methods.

2.2.47 FW_CRYPTO_KEY_EXCHANGE_TYPE

This enumeration is used to identify supported key exchange algorithms.

```
typedef  enum _tag_FW_CRYPTO_KEY_EXCHANGE_TYPE
{
  FW_CRYPTO_KEY_EXCHANGE_INVALID = 0,
  FW_CRYPTO_KEY_EXCHANGE_NONE = 0,
  FW_CRYPTO_KEY_EXCHANGE_DH1,
  FW_CRYPTO_KEY_EXCHANGE_DH2,
  FW_CRYPTO_KEY_EXCHANGE_ECDH256,
  FW_CRYPTO_KEY_EXCHANGE_ECDH384,
  FW_CRYPTO_KEY_EXCHANGE_DH2048,
  FW_CRYPTO_KEY_EXCHANGE_MAX
} FW_CRYPTO_KEY_EXCHANGE_TYPE;
```

FW_CRYPTO_KEY_EXCHANGE_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_CRYPTO_KEY_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_CRYPTO_KEY_EXCHANGE_DH1: Do key exchange with Diffie Hellman group 1. This symbolic constant has a value of 1.

FW_CRYPTO_KEY_EXCHANGE_DH2: Do key exchange with Diffie Hellman group 2. This symbolic constant has a value of 2.

FW_CRYPTO_KEY_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 385. This symbolic constant has a value of 4.

FW_CRYPTO_KEY_EXCHANGE_DH2048: Do key exchange with Diffie Hellman group 2,048. 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.

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

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

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

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

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

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

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

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

FW_CRYPTO_ENCRYPTION_AES_GCM256: Use 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 = FW_CRYPTO_ENCRYPTION_AES_GCM128 : 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.

2.2.49 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 only be used 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_CRYPTO_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_CRYPTO_HASH_MAX_V2_0 = FW_CRYPTO_HASH_SHA256: 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.

2.2.50 FW_CRYPTO_PROTOCOL_TYPE

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

```
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_ENCRYPTION_MAX
} FW_CRYPTO_PROTOCOL_TYPE;
```

FW_CRYPTO_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_CRYPTO_PROTOCOL_AH: Use the authentication header (AH) to enforce IPsec. This symbolic constant has a value of 1.

FW_CRYPTO_PROTOCOL_ESP: Use the ESP protocol header. This symbolic constant has a value of 2.

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

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 4.

2.2.51 FW_PHASE1_CRYPTO_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_CRYPTO_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.

2.2.52 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 validations checks that Phase 2 cryptographic suites must pass:

- **dwTimeoutMinutes** MUST be greater than or equal to 5 and less than or equal to 2,879.
- **dwTimeoutKBytes** 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_CRYPTO_PROTOCOL_AH or FW_CRYPTO_PROTOCOL_BOTH, the **AhHash** field MUST NOT be equal to FW_CRYPTO_HASH_NONE.
- If the **Protocol** field is FW_CRYPTO_PROTOCOL_BOTH, the **AhHash** field MUST be equal to the **EspHash** field.
- If the **Protocol** field is FW_CRYPTO_PROTOCOL_BOTH or FW_CRYPTO_PROTOCOL_ESP, **EspHash** MUST NOT be set to FW_CRYPTO_HASH_NONE, or **Encryption** MUST NOT be set to FW_CRYPTO_ENCRYPTION_NONE, but not both.

2.2.53 FW_PHASE1_CRYPTO_FLAGS

```
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;
```

FW_PHASE1_CRYPTO_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_CRYPTO_FLAGS_DO_NOT_SKIP_DH: This flag ensures that 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, roundtrips and the computational cost of DH are avoided.)

FW_PHASE1_CRYPTO_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.

2.2.54 FW_PHASE2_CRYPTO_PFS

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

```
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;
```

FW_PHASE2_CRYPTO_PFS_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_CRYPTO_PFS_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_CRYPTO_PFS_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_CRYPTO_PFS_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_CRYPTO_PFS_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_CRYPTO_PFS_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_CRYPTO_PFS_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_CRYPTO_PFS_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_CRYPTO_PFS_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.

2.2.55 FW_CRYPTO_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 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 CRYPTO SUITE pPhase1Suites;
      unsigned long dwTimeoutMinutes;
      unsigned long dwTimeoutSessions;
    };
    [case(FW_IPSEC_PHASE_2)]
      struct {
      FW PHASE2 CRYPTO PFS Pfs;
      [range(0,1000)] unsigned long dwNumPhase2Suites;
      [size is(dwNumPhase2Suites)] PFW PHASE2 CRYPTO SUITE 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_CRYPTO_SET,
 *PFW_CRYPTO_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 cryptographic set applies for Phase 1 (main mode) or Phase 2 (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 Phase 1 and with the "{E5A5D32A-4BCE-4e4d-B07F-4AB1BA7E5FE2}" string for Phase 2. 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_CRYPTO_FLAGS** enumeration bit flags.

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

pPhase1Suites: A pointer to an array of **dwNumPhase1Suites** contiguous **FW_PHASE1_CRYPTO_SUITE** 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_CRYPTO_PFS** 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 **dwNumPhase2Suites** contiguous **FW_PHASE2_CRYPTO_SUITE** elements.

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 representing the name of the originating GPO. It MUST be set if the origin is GP; otherwise, it MUST be null.

Status: Status code of the set, as specified by **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: Reserved value, and not currently used. It MUST be set to 0.

The following are semantic checks that cryptographic sets MUST pass:

- **wSchemaVersion** MUST NOT be less than 0x000200.
- **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 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** 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_CRYPTO_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 **dwNumPhase1Suites** field MUST agree with the **pPhase1Suites** field.

- 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 an **KeyExchange** field with the FW_CRYPTO_ENRYPTION_INVALID value, and MUST have valid values.
- If the set has a schema policy version of 0x0200, then all Phase1 suites MUST NOT have an **Encryption** field with values greater than or equal to FW_CRYPTO_ENCRYPTION_MAX_V2_0.
- All Phase1 suites MUST NOT have an **Encryption** field with the FW_CRYPTO_ENRYPTION_NONE value, and MUST have valid values less than FW_CRYPTO_ENCRYPTION_MAX_V2_0.
- If the set has a schema policy version of 0x0200 then, all Phase1 suites MUST NOT have a **Hash** field with values greater than or equal to FW_CRYPTO_HASH_MAX_V2_0.
- All Phase1 suites MUST NOT have a **Hash** field with the FW_CRYPTO_HASH_NONE value, and MUST have either MD5 (FW_CRYPTO_HASH_MD5) or SHA (FW_CRYPTO_HASH_SHA1, FW_CRYPTO _HASH_SHA256, FW_CRYPTO_HASH_SHA384)_valid values.

- If the **IpSecPhase** is FW_IPSEC_PHASE_2:
  - The **wszSetId** MUST NOT have the default Phase2 cryptographic set ID as a prefix.
  - The cryptographic set MUST have at least one Phase2 cryptographic suite.
  - The **dwNumSuites** field MUST agree with the **pSuites** field.
  - The **Pfs** field MUST NOT be FW_PHASE2_CRYPTO_PFS_INVALID, and MUST have valid values.
  - If the set has a schema policy version of 0x0200, then all Phase2 cryptographic suites MUST NOT have a **AhHash** field or **EspHash** field with values greater than or equal to FW_CRYPTO_HASH_MAX_V2_0.
  - If the set has a schema policy version of 0x0200, then all Phase2 suites MUST NOT have an **Encryption** field with values greater than or equal to FW_CRYPTO_ENCRYPTION_MAX_V2_0.
  - All Phase2 suites within the set MUST NOT have a **dwTimeoutMinutes** field less than FW_MIN_CRYPTO_PHASE2_TIMEOUT_MINUTES or greater than FW_MAX_CRYPTO_PHASE2_TIMEOUT_MINUTES.
  - All Phase2 suites within the set MUST NOT have a **dwTimeoutKBytes** field less than FW_MIN_CRYPTO_PHASE2_TIMEOUT_KBYTES or greater than FW_MAX_CRYPTO_PHASE2_TIMEOUT_KBYTES.
  - All the Phase2 suites within the set MUST NOT have a **Protocol** field with the FW_CRYPTO_PROTOCOL_INVALID, and MUST have valid values.
  - For all suites with the **Protocol** field equal to FW_CRYPTO_PROTOCOL_AH or to FW_CRYPTO_PROTOCOL_BOTH:
    - All suites MUST NOT have a **AhHash** field with the FW_CRYPTO_HASH_NONE value, and MUST have valid values not equal to FW_CRYPTO_HASH_SHA384.
  - For all suites with the **Protocol** field only equal to FW_CRYPTO_PROTOCOL_BOTH:
    - All suites MUST have the **AhHash** field equal to the **EspHash** field

- For all suites with the **Protocol** field equal to FW_CRYPTO_PROTOCOL_ESP:
  - All suites MUST have a **EspHash** field with valid values, including FW_CRYPTO_HASH_NONE. The **EspHash** field MUST NOT equal FW_CRYPTO_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 with the **Encryption** field equal to FW_CRYPTO_ENCRYPTION_AES_GCM128, 192 or 256 MUST also have a corresponding FW_CRYPTO_HASH_AES_GMAC128, 192or 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.

2.2.56 FW_BYTE_BLOB

```
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.

2.2.57 FW_COOKIE_PAIR

```
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.

2.2.58 FW_PHASE1_KEY_MODULE_TYPE

```
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 Auth IP.

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.

2.2.59 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 only use health certificate or certificate to account mapping flags, which represent certificate characteristics.

2.2.60 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 {
    [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;
```

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.

MyCert: This field contains the subject name and certification flags (health, account mapping, exclude ca) from the certificate of the local host that was used in the authentication process when a certificate-based authentication method is used.

PeerCert: This field contains the subject name and certification flags (health, account mapping, exclude ca) from the certificate of the remote host that was used in the authentication process when a certificate-based authentication method is used.

wszMyId: A pointer to a Unicode string representing the identity of the local host when a Kerberos-based, as specified in [MS-KILE], authentication method is used.

wszPeerId: A pointer to a Unicode string representing the identity of the remote host when a Kerberos-based, as specified in [MS-KILE], authentication method is used.

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

2.2.61 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.

2.2.62 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 number that uniquely identifies the security association.

KeyModuleType: The keying protocol used, IKE or AuthIP. The field MUST only contain 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 out of 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 out of 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.

2.2.63 FW_PHASE2_TRAFFIC_TYPE

This enumeration identifies the two types of traffic enforcement modes that IPsec supports.

```
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.

2.2.64 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. If the value is 0, it means any port.

wRemotePort: This field specifies the port of the remote endpoint that is used in the traffic secured by this security association. If the value is 0, it means 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.<5> 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 deemed appropriate.

dwP2SaFlags: Reserved value, and not currently used. It MUST be set to 0.

2.2.65 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:

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
dwNumLUIDs																															
pLUIDs																															
GUID1																															
…																															
…																															
…																															

On 64-bit servers:

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
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**.

2.2.66 FW_CONN_HANDLE

This type contains an RPC binding handle to an RPC interface implementing the Firewall and Advanced Security Protocol.

This type is declared as follows:

```
typedef handle_t FW_CONN_HANDLE;
```

2.2.67 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.

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. 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.

A store is composed of the following components: **global policy configuration options (section 2.2.36)**, **profile configuration options (section 2.2.32)**, 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 four types of policy objects: firewall rules, connection security rules, authentication sets, and crypto sets, which are defined in this protocol as the data types **FW_RULE (section 2.2.31)**, **FW_CS_RULE (section 2.2.42)**, **FW_AUTH_SET (section 2.2.46)**, and **FW_CRYPTO_SET (section 2.2.55)**, respectively. The connection security rules include references to authentication sets and crypto sets in the store to be able to 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. For more information on Windows choice of enforcement state objects, see [MSWFPSDK].<6>

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 in-between 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. 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.

Note that from this protocol's perspective, 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]; this information is not relevant to this protocol.

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 them 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 policy merged. While doing so, the server keeps enforcement state<7> in each of the linked nodes of rules objects and sets objects, if appropriate. 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 use of context handles created by a method of a different 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 serve 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 the exact 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 referenced by the specified opened policy store handle. Opnum: 5
RRPC_FWSetFirewallRule	This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol. Opnum: 6
RRPC_FWDeleteFirewallRule	This parameter is an RPC binding handle that connects to the RPC interface of the Firewall and Advanced Security Protocol. 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

Method	Description
RRPC_FWEnumFirewallRules	This 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. 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 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 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 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 referenced by the specified opened policy store handle. Opnum: 15
RRPC_FWEnumConnectionSecurityRules	This 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. Opnum: 16
RRPC_FWAddAuthenticationSet	This method requests the server to add the authentication set in the policy contained in the policy store referenced by the specified opened policy store handle. Opnum: 17

Method	Description
RRPC_FWSetAuthenticationSet	This method requests the server to modify the specified authentication set in the policy contained in the policy store 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 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 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 referenced by the *hPolicyStore* 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 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 referenced by the specified opened policy store handle. Opnum: 24
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 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 referenced by the *hPolicyStore* handle. The method returns a linked list of all these cryptographic objects. Opnum: 26

Method	Description
RRPC_FWEnumPhase1SAs	This method requests the server to return all the security associations of the IPsec first negotiation phase contained in the store referenced by the *hPolicyStore* 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 referenced by the *hPolicyStore* 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

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(W_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.

StoreType: This parameter specifies the policy store type the client wants to open.

AccessRight: This parameter specifies the read or read/write access rights 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.

phPolicyStore: This is an output parameter that provides a pointer to an **FW_POLICY_STORE_HANDLE** data type. If successful, this parameter will contain a handle to the opened store.

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].

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [MS-RPCE].

The server MUST validate the client credentials to the administrator or network operator before executing this method.

3.1.4.2 RRPC_FWClosePolicyStore (Opnum 1)

The **RRPC_FWClosePolicyStore** method receives an opened store handle and closes it, freeing any resources that were allocated by the server to serve operations on the opened store.

```
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 non-zero 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 the client credentials to the administrator or network operator 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 the exact same policy as 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 non-zero 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 the client credentials to the administrator or network operator 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 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.

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.
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: ▪ 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 the client credentials to the administrator or network operator 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 that the *pBuffer* parameter points to.

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 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: ▪ 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 null but *dwBufSize* 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 the client credentials to the administrator or network operator 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 referenced by the specified opened policy store handle.

```
unsigned long RRPC_FWAddFirewallRule(
  [in] FW_CONN_HANDLE rpcConnHandle,
  [in] FW_POLICY_STORE_HANDLE hPolicyStore,
  [in] PFW_RULE 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, successfully opened with the **RRPC_FWOpenPolicyStore (Opnum 0)** method. The handle MUST have read/write access rights.

pRule: This parameter represents the firewall rule 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.

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
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 *hPolicyStore* 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: ▪ The *pRule* 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

Return value/code	Description
	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 in disk. If called on an online store, the firewall rule is also enforced.

The server MUST validate the client credentials to the administrator or network operator 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 referenced by the specified opened policy store handle.

```
unsigned long RRPC_FWSetFirewallRule(
  [in] FW_CONN_HANDLE rpcConnHandle,
  [in] FW_POLICY_STORE_HANDLE hPolicyStore,
  [in] PFW_RULE 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, successfully opened with the **RRPC_FWOpenPolicyStore (Opnum 0)** method. The handle MUST have read/write access rights.

pRule: This parameter represents the firewall rule the client wants to modify in the store. The rule must be a valid rule, as specified in the definition of the **FW_RULE** data type.

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
0x00000032 ERROR_NOT_SUPPORTED	The specified store does not support this method; the store might be read-only.
0x00000005 ERROR_ACCESS_DENIED	The *hPolicyStore* 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.
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:

Return value/code	Description
	▪ The *pRule* 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 the client credentials to the administrator or network operator 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 specified opened policy store handle.

```
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 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
0x00000032 ERROR_NOT_SUPPORTED	The specified store does not support this method; the store might be read-only.
0x00000005 ERROR_ACCESS_DENIED	The *hPolicyStore* 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 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 the client credentials to the administrator or network operator 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 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.

The server MUST validate the client credentials to the administrator or network operator before executing this method.

```
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 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
0x00000032 ERROR_NOT_SUPPORTED	The specified store does not support this method; the store might be read-only.
0x00000005 ERROR_ACCESS_DENIED	The *hPolicyStore* 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].

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 referenced by the *hPolicyStore* handle. The method returns a linked list of all the firewall rule objects.

```
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_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 will use this bitmask to determine if rules should be returned or not. Rules that contain a status code of the class specified 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 will also use this parameter to determine if rules should be returned or not. 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**, 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 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 *hPolicyStore* 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 *dwProfileFilter* 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 the client credentials to the administrator or network operator 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.

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.
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: ▪ The store type specified does not support this method. ▪ The configuration option is not supported in this store. ▪ The *Profile* 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: ▪ 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 the client credentials to the administrator or network operator 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.

dwBufSize: This parameter is the size of the buffer that the *pBuffer* parameter points to.

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
0x00000032 ERROR_NOT_SUPPORTED	The method does not support the specified combination of parameters. This can be because: ▪ 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 *hPolicyStore* 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: ▪ 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 the configuration value. ▪ The buffer is null but *dwBufSize* 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 the client credentials to the administrator or network operator 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 referenced by the specified opened policy store handle.

```
unsigned long RRPC_FWAddConnectionSecurityRule(
  [in] FW_CONN_HANDLE rpcConnHandle,
  [in] FW_POLICY_STORE_HANDLE hPolicy,
  [in] PFW_CS_RULE 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, 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 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.

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
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 *hPolicyStore* 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: ▪ The *pRule* object did not pass the connection security rule validations specified in the definition

Return value/code	Description
	of the **FW_CS_RULE** 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 connection security rule in the connection security 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 connection security rule is also enforced.

The server MUST validate the client credentials to the administrator or network operator 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 referenced by the specified opened policy store handle.

```
unsigned long RRPC_FWSetConnectionSecurityRule(
  [in] FW_CONN_HANDLE rpcConnHandle,
  [in] FW_POLICY_STORE_HANDLE hPolicy,
  [in] PFW_CS_RULE 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, 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 the client wants to modify in the store. The rule must be a valid rule, as specified in the definition of the **FW_CS_RULE** data type.

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
0x00000032 ERROR_NOT_SUPPORTED	The specified store does not support this method; the store might be read-only.
0x00000005 ERROR_ACCESS_DENIED	The *hPolicyStore* 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	The specified rule referenced by the *wszRuleID* member string of

Return value/code	Description
ERROR_FILE_NOT_FOUND	the **FW_CS_RULE** 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: ▪ The *pRule* object did not pass the connection security rule validations specified in the definition of the **FW_CS_RULE** 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 connection security rule already stored in the connection security linked list of the memory representation of store being modified. It uses this list to determine if the rule exists or not. 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 the client credentials to the administrator or network operator 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 specified opened policy store handle.

```
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 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
0x00000032	The specified store does not support this method; the store might be

Return value/code	Description
ERROR_NOT_SUPPORTED	read-only.
0x00000005 ERROR_ACCESS_DENIED	The *hPolicyStore* 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 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 the client credentials to the administrator or network operator 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 specified opened policy store handle.

```
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 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
0x00000032 ERROR_NOT_SUPPORTED	The specified store does not support this method; the store might be read-only.
0x00000005 ERROR_ACCESS_DENIED	The *hPolicyStore* 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 the client credentials to the administrator or network operator 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 referenced by the *hPolicyStore* handle. The method returns a linked list of all the connection security rule objects.

```
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_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. 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 will use this bitmask to determine if rules should be returned or not. 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 will also use this parameter to determine if rules should be returned or not. 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**, 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 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.
0x00000057 ERROR_INVALID_PARAMETER	The *dwProfileFilter* 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 the client credentials to the administrator or network operator 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 specified opened policy store handle.

```
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 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
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 *hPolicyStore* 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	One of the parameters of this method is incorrect, or is required and not specified. This error can be returned because: ▪ The *pAuth* 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 the client credentials to the administrator or network operator 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 specified opened policy store handle.

```
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 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
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 *hPolicyStore* 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: ▪ The *pAuth* 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 the client credentials to the administrator or network operator 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 specified opened policy store handle.

```
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 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
0x00000962 ERROR_ACTIVE_CONNECTIONS	The specified set is still referenced by connection security rules. This failure only happens 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 *hPolicyStore* 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 the client credentials to the administrator or network operator 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 specified opened policy store handle.

```
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 this set is used in.

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
0x00000962 ERROR_ACTIVE_CONNECTIONS	The specified set is still referenced by connection security rules. This failure only happens 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 *hPolicyStore* 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 non-default set is referenced by a connection security rule.

The server MUST validate the client credentials to the administrator or network operator 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 *hPolicyStore* 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/write access rights.

IpSecPhase: This parameter specifies the specific IPsec negotiation phase this set applies to.

dwFilteredByStatus: This parameter is a combination of flags from the **FW_RULE_STATUS_CLASS** enumeration. This method will use 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**, which 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 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.
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: ▪ The *IpSecPhase* 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 the client credentials to the administrator or network operator 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 will not be valid for a **FW_AUTH_SET** (section 2.2.46) 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 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.

When the server is operating under normal security mode, the server MUST validate the client credentials to the administrator or network operator 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 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
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 *hPolicyStore* 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: ▪ The *pCrypto* 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].

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 specified opened policy store handle.

```
unsigned long RRPC_FWSetCryptoSet(
  [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 modify in 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 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
0x00000032 ERROR_NOT_SUPPORTED	The specified store does not support this method; the store might be read-only.
0x00000005 ERROR_ACCESS_DENIED	The *hPolicyStore* 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: ▪ The *pCrypto* 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 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.

When the server is operating under normal security mode, the server MUST validate the client credentials to the administrator or network operator 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 referenced by the specified opened policy store handle.

```
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
);
```

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 cryptographic set the client wants to delete from the specified store.

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
0x00000962 ERROR_ACTIVE_CONNECTIONS	The specified set is still referenced by connection security rules. This failure only happens 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 *hPolicyStore* 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	The specified IPsec phase is not a valid one.

Return value/code	Description
ERROR_INVALID_PARAMETER	

Exceptions Thrown: No exceptions are thrown beyond those 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 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.

When the server is operating under normal security mode, the server MUST validate the client credentials to the administrator or network operator 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.

3.1.4.26 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 referenced by the specified opened policy store handle.

```
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, 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.

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
0x00000962 ERROR_ACTIVE_CONNECTIONS	There are non-default sets still being referenced by connection security rules. There is always a default set to use, either from other stores or a hard-coded one; hence, 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	The *hPolicyStore* handle was not opened with read/write access

Return value/code	Description
ERROR_ACCESS_DENIED	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 beyond those 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 non-default set is referenced by a connection security rule.

When the server is operating under normal security mode, the server MUST validate the client credentials to the administrator or network operator 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.

3.1.4.27 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 *hPolicyStore* 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_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.

IpSecPhase: This parameter specifies the specific IPsec negotiation phase this set applies to.

dwFilteredByStatus: This parameter is a combination of flags from the **FW_RULE_STATUS_CLASS** enumeration. This method will use 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**, which 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 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.
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: ▪ The *IpSecPhase* 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 the client credentials to the administrator or network operator 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 will not be valid for a **FW_CRYPTO_SET** (section 2.2.55) 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 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 *hPolicyStore* 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 will return 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 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 the client credentials to the administrator or network operator 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 *hPolicyStore* 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 the client credentials to the administrator or network operator 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 will delete 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 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 the client credentials to the administrator or network operator 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 will delete 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 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 the client credentials to the administrator or network operator 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

No abstract data model is used.

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 timers 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

The following sections describe several operations as used in common scenarios to illustrate the function of the Firewall and Advanced Security Protocol.

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:

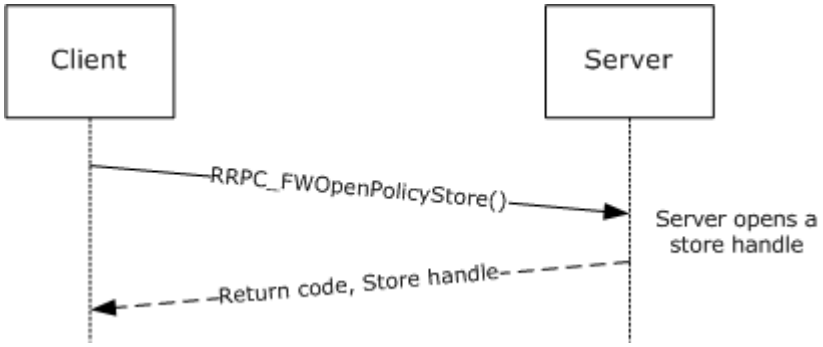

Figure 2: Opening a policy store

To open a policy store, the client must first get an rpcBinding to this interface in the server. Then it 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, it can perform operations on it. The protocol sequence that adds a firewall rule to the policy store is as follows:

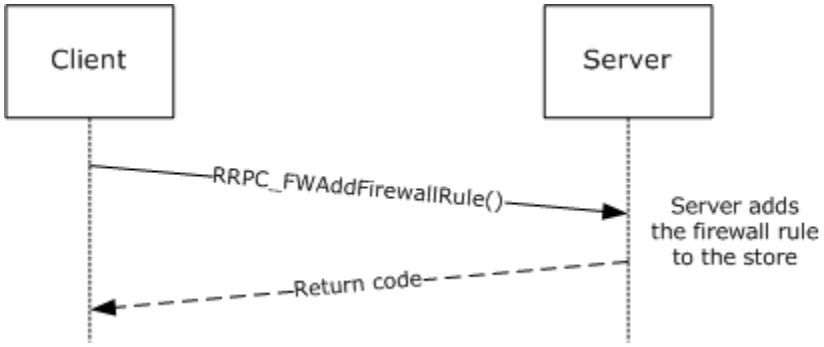

Figure 3: Adding a firewall rule

To add a firewall rule, the client application will first fill 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 with a different Rule Id, or bubble the error up.

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:

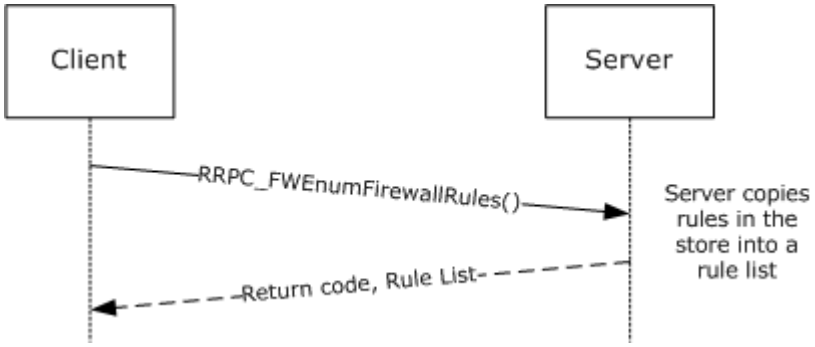

Figure 4: Enumerating firewall rules

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.

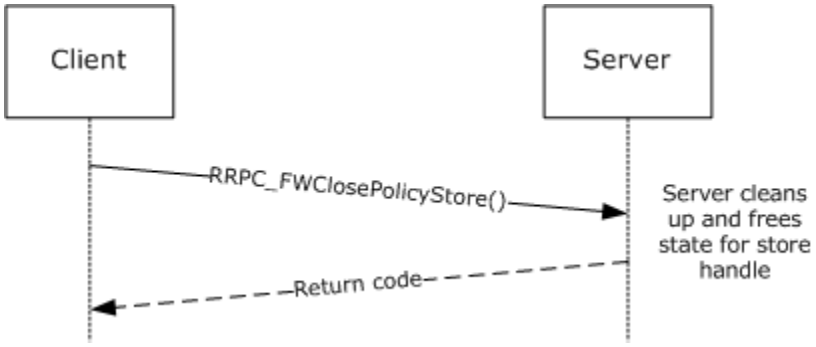

Figure 5: Closing a policy store

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

The following sections specify security considerations for implementers of the Firewall and Advanced Security Protocol.

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. For protection, the client must consider this value a good prediction. If the client must allocate a buffer based on the rule count, it can use this value to do it, but while filling it out, the client MUST actively validate that the actual objects linked do not overflow the object count. Failure to do this validation might result in buffer overruns in the client application code.

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_MAX               = 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_CRYPTO = 0x0008000A,
    FW_RULE_STATUS_PARSING_ERROR_PHASE2_AUTH   = 0x0008000B,
    FW RULE STATUS PARSING ERROR RESOLVE APP   = 0x0008000C,
    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 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 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 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 PHASE1 AUTH SET ID= 0x00100500,
FW RULE STATUS SEMANTIC ERROR PHASE2 CRYPTO SET ID = 0x00100510,

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 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 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 MACHINE USER AUTH = 0x00101070,

FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPTO_SET_ID  = 0x00105000,
FW RULE STATUS SEMANTIC ERROR PHASE1 CRYPTO FLAGS   = 0x00105001,
FW RULE STATUS SEMANTIC ERROR PHASE1 CRYPTO TIMEOUT MINUTES = 0x00105002,
FW RULE STATUS SEMANTIC ERROR PHASE1 CRYPTO TIMEOUT SESSIONS= 0x00105003,
FW RULE STATUS SEMANTIC ERROR PHASE1 CRYPTO KEY EXCHANGE    = 0x00105004,
FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPTO_ENCRYPTION      = 0x00105005,
FW_RULE_STATUS_SEMANTIC_ERROR_PHASE1_CRYPTO_HASH            = 0x00105006,
FW RULE STATUS SEMANTIC ERROR PHASE1 CRYPTO ENCRYPTION VER  = 0x00105007,
FW RULE STATUS SEMANTIC ERROR PHASE1 CRYPTO HASH VER        = 0x00105008,


FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPTO_PFS             = 0x00105020,
FW_RULE_STATUS_SEMANTIC_ERROR_PHASE2_CRYPTO_PROTOCOL        = 0x00105021,
FW RULE STATUS SEMANTIC ERROR PHASE2 CRYPTO ENCRYPTION      = 0x00105022,
FW RULE STATUS SEMANTIC ERROR PHASE2 CRYPTO 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_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              = 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 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_MAX                  = 0x0040
}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 = 0x00010,
    FW RULE FLAGS MAX               = 0x0020,
}FW_RULE_FLAGS;


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;

} 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 MAX            =  0x0008,
    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 MAX
} FW_GLOBAL_CONFIG;

typedef enum  FW CONFIG FLAGS
{
    FW CONFIG FLAG RETURN DEFAULT IF NOT FOUND  = 0x0001




} FW CONFIG FLAGS;


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 MAX             = 0x02
}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 RULE
{
    struct _tag_FW_CS_RULE *pNext;
    unsigned short               wSchemaVersion;
    [string, range(1,1001), 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 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                                 = 0x0020
}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*              wszGPOName;
    FW_RULE_STATUS      Status;
    unsigned long               dwAuthSetFlags;
}FW AUTH SET, *PFW AUTH SET;


typedef enum  tag FW CRYPTO KEY EXCHANGE TYPE
{
    FW_CRYPTO_KEY_EXCHANGE_INVALID = 0,
    FW CRYPTO KEY EXCHANGE NONE = 0,
    FW CRYPTO KEY EXCHANGE DH1,
    FW CRYPTO KEY EXCHANGE DH2,
    FW_CRYPTO_KEY_EXCHANGE_ECDH256,
    FW_CRYPTO_KEY_EXCHANGE_ECDH384,
    FW CRYPTO KEY EXCHANGE DH2048,
    FW CRYPTO KEY EXCHANGE MAX
}FW CRYPTO KEY 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 MAX
}FW CRYPTO PROTOCOL TYPE;



typedef struct _tag_FW_PHASE1_CRYPTO_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;



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 CRYPTO SUITE   pPhase1Suites;
            unsigned long                     dwTimeoutMinutes;
            unsigned long                     dwTimeoutSessions;
        };
        [case(FW_IPSEC_PHASE_2)]
        struct
        {
            FW PHASE2 CRYPTO PFS      Pfs;
            [range(0, 1000)]
            unsigned long                     dwNumPhase2Suites;
            [size_is(dwNumPhase2Suites)]
            PFW PHASE2 CRYPTO SUITE   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 CRYPTO SET, *PFW CRYPTO SET;

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_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;


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;


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(6b5bdd1e-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;




    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_RULE           pRule
    );


unsigned long
RRPC_FWSetFirewallRule(
    [in] FW CONN HANDLE          rpcConnHandle,
    [in] FW POLICY STORE HANDLE  hPolicyStore,
    [in] PFW RULE           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_RULE         *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 RULE           pRule
    );


unsigned long
RRPC_FWSetConnectionSecurityRule(
    [in] FW CONN HANDLE             rpcConnHandle,
    [in] FW POLICY STORE HANDLE     hPolicy,
    [in] PFW CS RULE           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_RULE*      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 CRYPTO SET     pCrypto
    );


unsigned long
RRPC FWSetCryptoSet(
    [in] FW CONN HANDLE          rpcConnHandle,
    [in] FW POLICY STORE HANDLE  hPolicy,
    [in] PFW_CRYPTO_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 CRYPTO SET*      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
        );


}
```

7 Appendix B: Windows Behavior

The information in this specification is applicable to the following versions of Windows:

- Windows Vista
- Windows Server 2008
- Windows Vista SP1

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies Windows behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that Windows does not follow the prescription.

<1> Section 1.7: Windows Vista uses version 0x0200. Windows Vista SP1 and Windows Server 2008 use version 0x0201.

<2> Section 2.2.25: 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 2.2.31)** and **FW_CS_RULE (section 2.2.42)** rules.

<3> Section 2.2.27: The Windows Vista firewall and advanced security component has specific hardcoded objects for default Phase 1 and Phase 2 authentication (see section 2.2.46) and cryptographic (see section 2.2.55) 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_CRYPTO_SET_NAME_STR
                        L"Service Hardcoded Default Phase1 CryptoSet"
#define FW_DEFAULT_P2_CRYPTO_SET_NAME_STR
                        L"Service Hardcoded Default Phase2 CryptoSet"

FW_PHASE1_CRYPTO_SUITE g_DefaultCryptoSuitesPhase1[] =
{
    {FW_CRYPTO_KEY_EXCHANGE_DH2,
     FW_CRYPTO_ENCRYPTION_AES128,
     FW_CRYPTO_HASH_SHA1},
    {FW_CRYPTO_KEY_EXCHANGE_DH2,
     FW_CRYPTO_ENCRYPTION_3DES,
     FW_CRYPTO_HASH_SHA1}
};

FW_CRYPTO_SET g_DefaultCryptoSetPhase1 =
{
    NULL,
    0x0200,
    FW_IPSEC_PHASE_1,
    FW_PHASE1_CRYPTO_SET,  // Notice: it should be called default
                           // phase1 crypto set
    FW_DEFAULT_P1_CRYPTO_SET_NAME_STR,
    FW_DEFAULT_P1_CRYPTO_SET_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_CRYPTO_PFS_DISABLE;
    g_DefaultCryptoSetPhase2.dwNumPhase2Suites =
                        RTL_NUMBER_OF(g_DefaultCryptoSuitesPhase2);
    g_DefaultCryptoSetPhase2.pPhase2Suites =
                        g_DefaultCryptoSuitesPhase2;
}
```

<4> Section 2.2.37: 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.

<5> Section 2.2.64: Windows Vista chooses enforcement state objects to be filter handles provided by the Windows Filtering Platform (for more information, see [MSWFPSDK]).

<6> 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]).

<7> 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]).

8 Index

L

M

N

O

P

R

S

T

V

W