[MS-TSGU]: Terminal Services Gateway Server Protocol Specification

Revision Summary

Date	Revision History	Revision Class	Comments
02/22/2007	0.01		MCPP Milestone 3 Initial Availability
06/01/2007	1.0	Major	Updated and revised the technical content.
07/03/2007	1.0.1	Editorial	Revised and edited the technical content.
07/20/2007	1.1	Minor	Updated the technical content.
08/10/2007	2.0	Major	Updated and revised the technical content.
09/28/2007	3.0	Major	Updated and revised the technical content.
10/23/2007	4.0	Major	Updated and revised the technical content.
11/30/2007	4.0.1	Editorial	Revised and edited the technical content.
01/25/2008	5.0	Major	Updated and revised the technical content.
03/14/2008	6.0	Major	Updated and revised the technical content.
05/16/2008	6.0.1	Editorial	Revised and edited the technical content.
06/20/2008	6.0.2	Editorial	Revised and edited the technical content.
07/25/2008	6.0.3	Editorial	Revised and edited the technical content.
08/29/2008	7.0	Major	Updated and revised the technical content.
10/24/2008	8.0	Major	Updated and revised the technical content.
12/05/2008	9.0	Major	Updated and revised the technical content.
01/16/2009	10.0	Major	Updated and revised the technical content.
02/27/2009	11.0	Major	Updated and revised the technical content.
04/10/2009	12.0	Major	Updated and revised the technical content.
05/22/2009	13.0	Major	Updated and revised the technical content.
07/02/2009	14.0	Major	Updated and revised the technical content.
08/14/2009	15.0	Major	Updated and revised the technical content.
09/25/2009	16.0	Major	Updated and revised the technical content.
11/06/2009	17.0	Major	Updated and revised the technical content.
12/18/2009	18.0	Major	Updated and revised the technical content.
01/29/2010	19.0	Major	Updated and revised the technical content.

Date	Revision History	Revision Class	Comments
03/12/2010	20.0	Major	Updated and revised the technical content.
04/23/2010	21.0	Major	Updated and revised the technical content.
06/04/2010	22.0	Major	Updated and revised the technical content.
07/16/2010	23.0	Major	Significantly changed the technical content.
08/27/2010	24.0	Major	Significantly changed the technical content.
10/08/2010	25.0	Major	Significantly changed the technical content.
11/19/2010	25.0	No change	No changes to the meaning, language, or formatting of the technical content.
01/07/2011	25.0	No change	No changes to the meaning, language, or formatting of the technical content.
02/11/2011	26.0	Major	Significantly changed the technical content.

Contents

1 Introduction

The Terminal Services Gateway Server Protocol [MS-TSGU] is a remote procedure call (RPC) protocol using HTTP as the transport mechanism, as specified in [C706] and extended in [MS-RPCE] and [MS-RPCH]. The Terminal Services Gateway Server Protocol is used primarily for tunneling client to server traffic across firewalls when the Terminal Services Gateway (TSG) server is deployed in the neutral zone of a network. The primary consumer of the Terminal Services Gateway Server Protocol is Remote Desktop Protocol: Basic Connectivity and Graphics Remoting Specification.

1.1 Glossary

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

authentication level
Authentication Service (AS)
certificate
client
endpoint
globally unique identifier (GUID)
handle
HRESULT
Hypertext Transfer Protocol over Secure Sockets Layer (HTTPS)
Interface Definition Language (IDL)
Internet Protocol version 4 (IPv4)
Internet Protocol version 6 (IPv6)
Network Data Representation (NDR)
opnum
Remote Desktop Protocol (RDP)
remote procedure call (RPC)
server (1)
SHA-1 hash
statement of health (SoH)
statement of health response (SoHR)
Triple Data Encryption Standard
Unicode
universally unique identifier (UUID)
well-known endpoint

The following terms are specific to this document:

administrative message: A message sent by the TS Gateway administrator to all users connected through TS Gateway. Typical would be messages sent for maintenance downtimes. The term Administrative Message and Service Message is used interchangeably in this document.

channel: A term indicating a successful connection between the TSG **client** and **target server** via the TSG **server**. For more information, see Data Transfer Phase (section 1.3.1.2).

Consent Signing Message: This is a EULA which the user must accept in order to connect successfully though TS Gateway.

Network Access Protection (NAP): A technology used to reduce the security risks associated with allowing external clients to connect to the network. It is implemented through quarantines and health checks, as specified in [MS-SOH].

out pipe: See pipe.

pipe: A supported IDL data type for streaming data, as specified in [C706] section 4.2.14. The term out pipe refers to the pipe created between the **client** and the TSG server for transferring data from the target server to the **client** via the TSG server. The term out pipe is used because the data flows from TSG server to **client**.

pluggable authentication: There is an option to override the default **RPC authentication** schemes by cookie-based authentication. TSG will load an installed plugin which will do the authentication based on a cookie which is passed by the client. The cookie will be retrieved when user browses a given site and enters their credentials.

RPC authentication: **RPC** supports several authentication methods as defined in [MS-RPCE] sections 1.7 and 2.2.1.1.7. Out of these, TSG **server** supports NTLM and **Schannel** authentication methods.

re-authentication: A process to validate the user credential's user authorization after the connection is established. This gives the provision to check the validity of user credentials and user authorization periodically, and disconnect the connection if the user credentials become invalid. In the process of **re-authentication**, the TS Gateway server expects the client to follow the same sequence of connection setup phase steps, as specified in section 1.3.1.1, to enable the credentials of the user to be re-checked, or **re-authenticated**. If the same sequence of steps is not followed, or an error occurs during the process, the existing connection is disconnected.

Schannel: Secure channel (**Schannel**) is an authentication method which can be used with **RPC authentication** by using RPC_C_AUTHN_GSS_SCHANNEL security provider as defined in [MS-RPCE] section 2.2.1.1.7.

target server: The resource that the client connects to via TSG server. The target server name is the machine name of such a resource. See sections 3.1.1 and 3.2.1 for more information on the Abstract Data Model (ADM) element **Target server name**.

tunnel: A tunnel establishes a context in which all further method calls or data transfer can be performed between the TSG **client** and the TSG **server**. A tunnel is unique to a given combination of a TSG **server** and TSG **client** instance. All operations on the tunnel are stateful.

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-DTYP] Microsoft Corporation, "Windows Data Types", January 2007.

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

[MS-NAPSO] Microsoft Corporation, "Network Policy and Access Services System Overview", August 2009.

[MS-RDPBCGR] Microsoft Corporation, "Remote Desktop Protocol: Basic Connectivity and Graphics Remoting Specification", June 2007.

[MS-RNAP] Microsoft Corporation, "Vendor-Specific RADIUS Attributes for Network Access Protection (NAP) Data Structure", January 2007.

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

[MS-RPCH] Microsoft Corporation, "Remote Procedure Call over HTTP Protocol Specification", January 2007.

[MS-SOH] Microsoft Corporation, "Statement of Health for Network Access Protection (NAP) Protocol Specification", January 2007.

[MS-TSSO] Microsoft Corporation, "Terminal Services System Overview", February 2009.

[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

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

[MSDN-MMSCH] Microsoft Corporation, "Mixed Mode Serialization of Context Handles", http://msdn.microsoft.com/en-us/library/aa367098(VS.85).aspx

[MSDN-NAPAPI] Microsoft Corporation, "NAP Interfaces", http://msdn.microsoft.com/en-us/library/aa369705(v=VS.85).aspx

[MSDN-RPCMESSAGE] Microsoft Corporation, "RPC_MESSAGE", http://msdn.microsoft.com/en-us/library/aa378631.aspx

1.3 Overview

The Terminal Services Gateway Server Protocol is based on the Remote Procedure Call Over HTTP Protocol. The Terminal Services Gateway Server Protocol is designed for remote connections from Terminal Services Gateway (TSG) **clients** originating on the Internet to **target servers** behind a firewall.<1>

This protocol establishes a connection from a TSG client to a TSG **server** in the neutral zone. This connection is called a **tunnel** by the Terminal Services Gateway Server Protocol.

The TSG client then uses the tunnel to establish a **channel** between the TSG client and the target server with the TSG server acting as a proxy. Data transfer between the TSG client and the target server occurs by using the channel. The tunnel and channel maintain active connections.

Communication from the TSG server to the TSG client occurs by using an **RPC out pipe**. Communication from the TSG client to the TSG server occurs by using RPC calls. The TSG client calls **TsProxyCreateTunnel**, **TsProxyAuthorizeTunnel**, and **TsProxyCreateChannel** in the sequential order as shown in Figure 1—the TSG client calls the next call only after a response for the already issued call is received. Next, the TSG client calls **TsProxySetupReceivePipe** and **TsProxySendToServer** only after the **TsProxyCreateChannel** call has successfully completed. However, the **TsProxySetupReceivePipe** call may have multiple responses from the TSG server and these may be interspersed with **TsProxySendToServer**. To end the connection, the TSG client calls **TsProxyCloseChannel** and **TsProxyCloseTunnel** in the sequential order shown in the diagrams in the subsequent sections. If the TSG client calls the **TsProxyCloseTunnel** method before calling the **TsProxyCloseChannel** method, the TSG server closes the channel and then closes the tunnel. If **TsProxyCloseChannel** is called after **TsProxyCloseTunnel**, the TSG client receives an RPC exception. Refer to HRESULT Return Codes (section 2.2.2.24) for details on the errors returned.

1.3.1 RPC Call Phases

The Terminal Services Gateway Server Protocol operates in three different phases: a connection setup phase, a data transfer phase, and a shutdown phase. The following sections describe these phases.

1.3.1.1 Connection Setup Phase

During this phase, a connection between the TSG client and TSG server is established. The TSG server in turn establishes a connection to the target server. It consists of the following operations:

- Tunnel creation: Involves negotiating protocol versioning and capabilities, returning the server **certificate**, and returning a context representation for the tunnel to the TSG client. The TSG client can then present the context representation to the TSG server in subsequent operations on the tunnel. Tunnel creation is accomplished by using the **TsProxyCreateTunnel** method call. This is always the first call in the protocol sequence. A tunnel shutdown, as specified in section 3.1.4.3.3, is possible without proceeding further in the TSG protocol sequence.

- Tunnel authorization: Can involve performing any authorizing rules for the TSG client connection, health checks, quarantine, enforcing user authentication, performing health remediation if needed, and terminal server device redirection settings. This is accomplished by using a call to the **TsProxyAuthorizeTunnel** method. This is the second call in the protocol sequence. A tunnel shutdown is also possible after tunnel authorization.

- Request for messages: After the tunnel is authorized, if the client and the server are both capable of sending and receiving administrative messages, the TSG client can call **TsProxyMakeTunnelCall (section 3.1.4.1.3)**, with TSG_TUNNEL_CALL_ASYNC_MSG_REQUEST as the parameter. When the server has a message to send over to the client, the server completes this pending call. The client should then make another call to TsProxyMakeTunnelCall.

- Channel creation: Requires making a connection to the target server and may include access checks on whether a connection is allowed. A channel creation involves creating a server context representation for the channel and returning the context representation to the TSG client. The TSG client can then present the context representation in subsequent operations on the channel. This is accomplished by using the **TsProxyCreateChannel** method call. This is the third call in the protocol sequence. A channel shutdown, as specified in section 3.1.4.3.1, is possible without proceeding further in the TSG protocol sequence. A tunnel shutdown is only possible after all channels inside the tunnels are shut down. While TSG client SHOULD close the channels before requesting tunnel shutdown, the channel closure will be done automatically by the server when not already effected by the client.

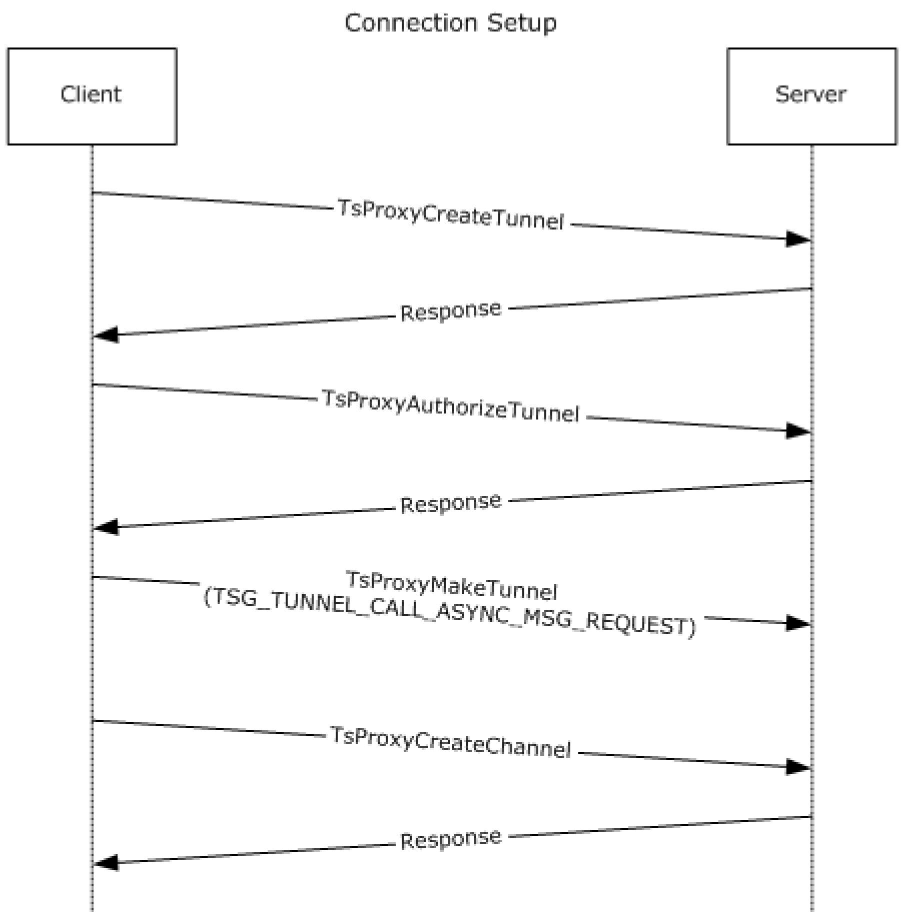

Figure 1: Method call sequence between the TSG client and TSG server during Connection Setup Phase

1.3.1.2 Data Transfer Phase

This phase allows for data transfer between the TSG client and the **target server** via the TSG server.

- Data transfer from the target server to the TSG client via the TSG server using an out pipe: The data from the target server is sent by the TSG server to the TSG client via the out pipe. In order to stream data from the TSG server to the TSG client, the Terminal Services Gateway Server Protocol utilizes RPC out pipes. All the data is streamed via this **pipe**. The out pipe setup involves creating an RPC out pipe. The out pipe setup is accomplished via a call to the **TsProxySetupReceivePipe** method. This is the fourth call in the protocol sequence; it can be called only once per channel. The **TsProxySetupReceivePipe** is called once, but data is sent from the TSG server to TSG client multiple times.

- Data transfer from the TSG client to the target server via the TSG server using an RPC call: The TSG client uses an RPC method call to send the data that is intended to be delivered to the target server by the TSG server. Data transfer from the TSG client to the TSG server is accomplished by using an RPC method call. The method call transfers data from the TSG client to the TSG server which then sends this data to the target server. The return value indicates success or failure. This is accomplished by using a call to the **TsProxySendToServer** method. This is the fifth call in the protocol sequence; it can be called multiple times within a channel.

- The TSG server acts as a proxy between the TSG client and the target server, as shown in the following diagram.

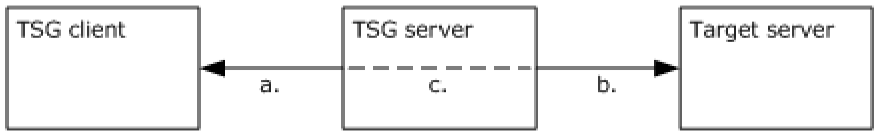

Figure 2: Connection between the TSG client and the target server.

The TSG client establishes a connection to the TSG server. The TSG server establishes a separate connection to the target server. Thus, a channel is a logical connection between the TSG client and the target server via the TSG server.

A channel can only be established within the context of a tunnel. The channel is specific to the TSG client and tunnel instance. Multiple channels can exist within a tunnel.

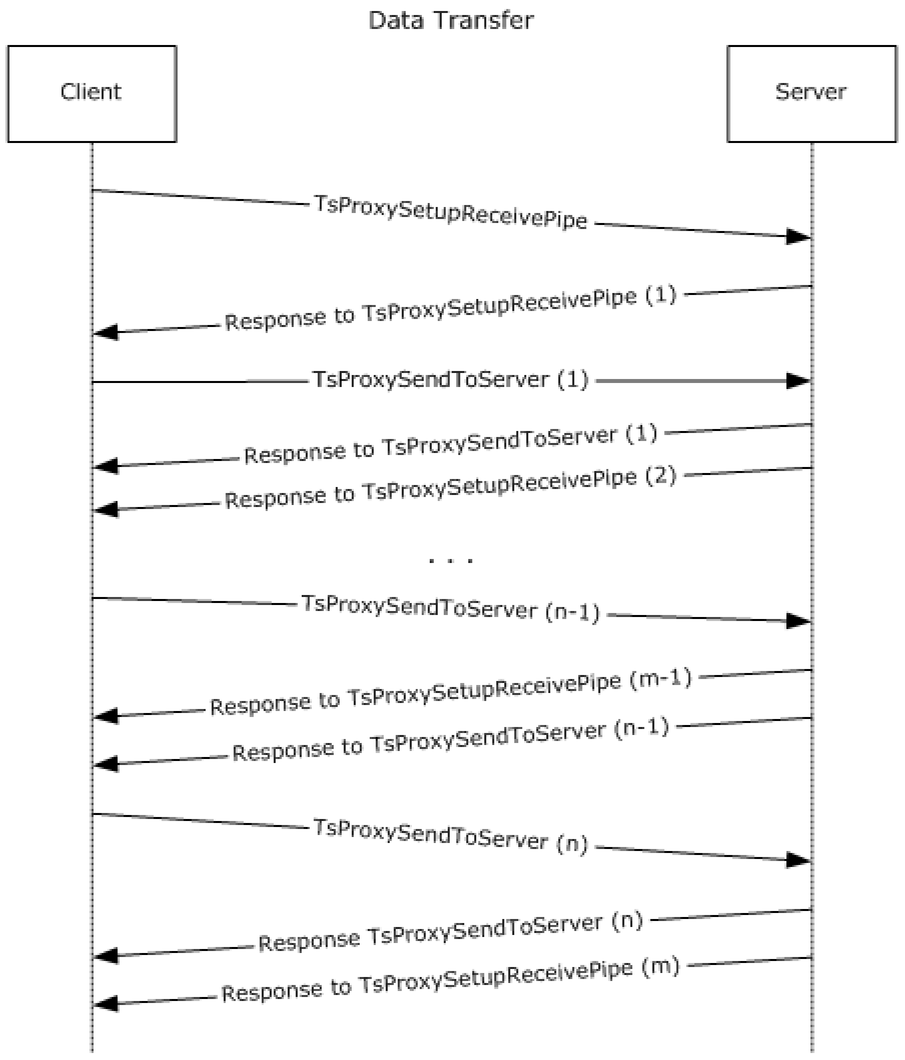
Data Transfer
Client
Server
TsProxySetupReceivePipe
Response to TsProxySetupReceivePipe (1)
TsProxySendToServer (1)
Response to TsProxySendToServer (1)
Response to TsProxySetupReceivePipe (2)
. . .
TsProxySendToServer (n-1)
Response to TsProxySetupReceivePipe (m-1)
Response to TsProxySendToServer (n-1)
TsProxySendToServer (n)
Response TsProxySendToServer (n)
Response to TsProxySetupReceivePipe (m)

Figure 3: Method call sequence between the TSG client and TSG server during Data Transfer Phase

1.3.1.3 Shutdown Phase

This phase is used to terminate the channel and tunnel.

- Channel shutdown: Channel shutdown can be performed only after a successful channel creation. A channel shutdown closes the RPC out pipe created in the data transfer phase and prevents any further use of the channel. The closing of a channel can be initiated either by the client or the TSG server. To initiate channel shutdown the client uses the **TsProxyCloseChannel** method call. The TSG server initiates channel shutdown sending an RPC response PDU with the PFC_LAST_FRAG bit set in the **pfc_flags** field as the final response PDU of the **TsProxySetupReceivePipe** method, as specified in section 3.1.4.2.2.
- Cancel pending messages: If the client has any pending administrative message request on the TSG server, the client cancels the same by making a **TsProxyMakeTunnel** call with **TSG_TUNNEL_CANCEL_ASYNC_MSG_REQUEST** as a parameter.

- Tunnel shutdown: Tunnel shutdown can be performed only after a successful tunnel creation and after all channels (if any) inside the tunnel are shut down successfully. A tunnel shutdown closes the connection between the TSG client and TSG server. This is the last call in the protocol sequence. The closing of a tunnel is accomplished by using the **TsProxyCloseTunnel** method call.

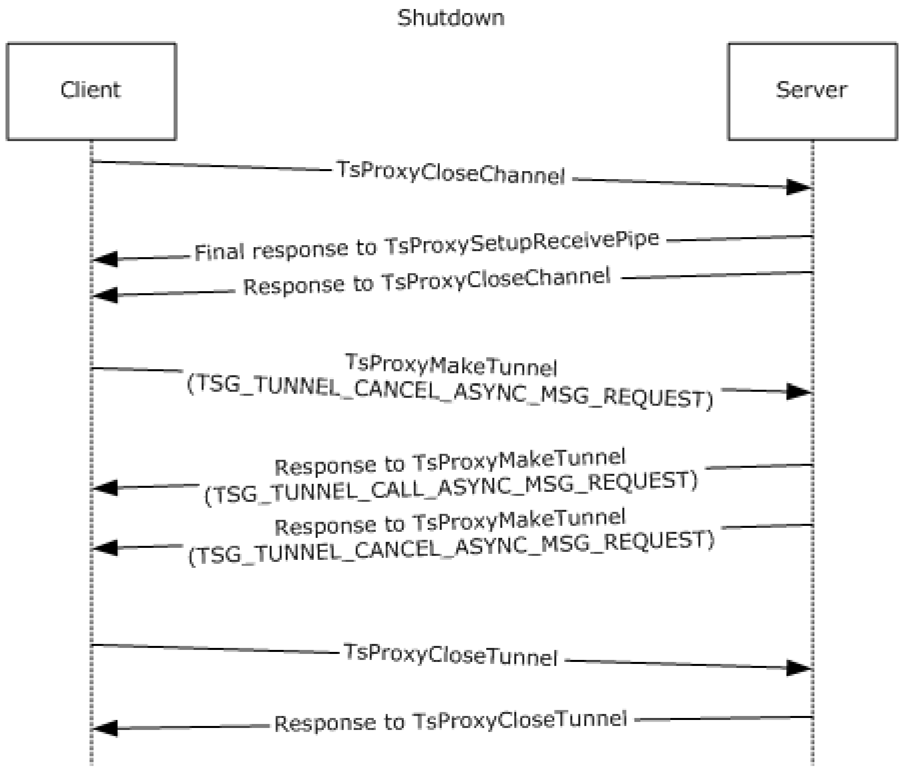
Shutdown
Client
Server
TsProxyCloseChannel
Final response to TsProxySetupReceivePipe
Response to TsProxyCloseChannel
TsProxyMakeTunnel (TSG_TUNNEL_CANCEL_ASYNC_MSG_REQUEST)
Response to TsProxyMakeTunnel (TSG_TUNNEL_CALL_ASYNC_MSG_REQUEST)
Response to TsProxyMakeTunnel (TSG_TUNNEL_CANCEL_ASYNC_MSG_REQUEST)
TsProxyCloseTunnel
Response to TsProxyCloseTunnel

Figure 4: Method call sequence between the TSG client and TSG server during Shutdown Phase

1.4 Relationship to Other Protocols

This protocol is dependent upon [MS-RPCH] for its transport.

No other protocol currently depends on the Terminal Services Gateway Server Protocol.

The **RDP** client and target server can use the Terminal Services Gateway Server Protocol as its transport for traversing corporate firewalls. RDP data is passed through this transport. Therefore, RDP is not aware of the TSG protocol. RDP is specified in [MS-RDPBCGR].

1.5 Prerequisites/Preconditions

This protocol is a Remote Procedure Call Over HTTP Protocol type interface and therefore has the prerequisites specified in [C706] parts 2, 3, and 4, [MS-RPCE] sections 2 and 3, and [MS-RPCH] section 2.1.

It is assumed that a TSG client has obtained the name of the TSG server that supports the TSG service before this protocol is invoked.

It is also assumed that a TSG client has obtained the name of the target server for making a channel connection.

If **HTTPS** transport is used, a certificate must be deployed on the TSG server. The root authority of the certificate must be trusted on the client as required by HTTPS.

1.6 Applicability Statement

This protocol is applicable when a client on the Internet or local private network requires a connection to a target server that is behind a firewall.

1.7 Versioning and Capability Negotiation

- Supported Transports: This protocol uses [MS-RPCH] as its only supported transport.
- Protocol Version: The Terminal Services Gateway Server Protocol RPC interface has a single version number of 1.3. The Terminal Services Gateway Server Protocol may be extended without altering the version number by adding RPC methods to the interface with **opnums** lying numerically beyond those defined in this specification. A TSG client determines whether such methods are supported by attempting to invoke the method; if the method is not supported, the TSG server returns an RPC_S_PROCNUM_OUT_OF_RANGE error. RPC versioning and capacity negotiation is specified in [C706] section 4.2.4.2 and [MS-RPCE] section 1.7. The **NDR** version required for this transport is 0x50002.
- Security and Authentication Methods: The Terminal Services Gateway Server Protocol supports all of the authentication methods as specified in [MS-RPCE] section 1.7. NTLM and Schannel are two of the authentication methods that have pluggable security provider modules, as specified in [MS-RPCE] section 2.2.1.1.7. **RPC authentication** APIs are also specified in [C706] section 2.7. In addition to RPC authentication, the TSG protocol supports cookie-based **pluggable authentication**.

  Also, Terminal Services Gateway Server Protocol does not make NTLM, Schannel, and Basic authentication calls directly but uses RPC over HTTP instead. RPC/HTTP (MS_RPCH) is pointed out in [MS-RPCE] section 2.1.1.8 RPC over HTTP (ncacn_http). The NTLM sequence for RPC is in section 4.2.
- Capability Negotiation: This protocol does not enforce any explicit version negotiation, but there is support for version negotiation. There is an explicit capabilities check done by the TSG client to ensure that its capabilities are supported and matched by the TSG server. The TSG client and TSG server announce their version and capabilities by using the **TsProxyCreateTunnel** method call. For specifications on the current version and capabilities announced by the TSG client and TSG server, see section 2.2.3.

1.8 Vendor-Extensible Fields

This protocol uses **HRESULT** datatypes as specified in [MS-ERREF] section 2.1. 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.

1.9 Standards Assignments

Parameter	Value	Reference
RPC interface UUID	44e265dd-7daf-42cd-8560-3cdb6e7a2729	[C706] section 2.3

Parameter	Value	Reference
endpoint	80, 443 and 3388	Section 2.1
ProtocolSequence	ncacn_http	Section 1.5

2 Messages

The following sections specify how the Terminal Services Gateway Server Protocol messages are transported and common data types.

2.1 Transport

This protocol uses the Remote Procedure Call over HTTP Protocol [MS-RPCH] as transport.

This protocol uses the following static **endpoints** as well as **well-known endpoints**. These endpoints are ports for [MS-RPCH] section 1.5 on the TSG server. The only protocol sequence used for the transport is "ncacn_http".

- Port 80: This endpoint is used by [MS-RPCH] as the underlying transport, when [MS-RPCH] runs over plain HTTP.
- Port 443: This endpoint is used by [MS-RPCH] as the underlying transport, when [MS-RPCH] runs over HTTPS.
- Port 3388: This endpoint is used by the TSG server to listen for incoming RPC method calls. The authenticated RPC interface allows RPC to negotiate the use of authentication and the **authentication level** on behalf of the TSG client and target server.

Port 3388 endpoint and at least one of Port 80 and Port 443 endpoints MUST be supported.

The Terminal Services Gateway Server Protocol MUST use the **UUID**, as specified in section 1.9. The RPC version number is 1.3.

2.2 Common Data Types

In addition to the RPC base types and definitions as specified in [C706] section 3.1, [MS-RPCE] section 2.2 and [MS-DTYP], additional data types are defined below in section 2.2.1.

In addition to the RPC base types and definitions described, the additional data types given below are defined in the MIDL specification for this RPC interface.

2.2.1 Data Types

2.2.1.1 RESOURCENAME

This type is declared as follows:

```
typedef [string] wchar_t* RESOURCENAME;
```

The target server name to which the TSG server connects. This refers to the ADM element **Target server name** (sections 3.1.1 and 3.2.1). The name MUST not be NULL and SHOULD be a valid server name. A valid target server name is one which DNS can resolve properly. Also, a valid target server is one which should be up and running and can accept a terminal server connection.

A **RESOURCENAME** can be an IP address, FQDN, or NetBIOS name. It may be noted that DNS cannot resolve all NetBIOS names - for example, there are differences in the allowed characters, differences in length, and differences in composition rules. Therefore, RESOURCENAME can be a

NetBIOS name if the NetBIOS name uses characters and length restrictions that DNS allows for, allowing DNS to resolve the name.

2.2.1.2 PTUNNEL_CONTEXT_HANDLE_NOSERIALIZE

An RPC context **handle** representing the tunnel for the given connection. Refer to [MSDN-MMSCH] for details on the modes of the context handles. For the NOSERIALIZE context handle, there may be more than one pending RPC call on the TSG server. However, on the wire it is identical to PTUNNEL_CONTEXT_HANDLE_SERIALIZE.

This type is declared as follows:

```
typedef [context_handle] void* PTUNNEL_CONTEXT_HANDLE_NOSERIALIZE;
```

The context handle MUST NOT be type_strict, but it MUST be strict. More details on RPC context handles are specified in [C706] sections 4.2.16.6, 5.1.6, and 6.1 and [MS-RPCE] section 3.1.1.5.3.2.2.2 and 3.3.1.4.1.

2.2.1.3 PCHANNEL_CONTEXT_HANDLE_NOSERIALIZE

An RPC context handle representing the channel for the given connection. Refer to [MSDN-MMSCH] for details on modes of the context handles. For the NOSERIALIZE context handle, there may be more than one pending RPC call on the TSG server. However, on the wire it is identical to PCHANNEL_CONTEXT_HANDLE_SERIALIZE.

This type is declared as follows:

```
typedef [context_handle] void* PCHANNEL_CONTEXT_HANDLE_NOSERIALIZE;
```

The context handle MUST NOT be type_strict, but it MUST be strict. More details on RPC context handles are specified in [C706] sections 4.2.16.6, 5.1.6, and 6.1 and [MS-RPCE] section 3.1.1.5.3.2.2.2 and 3.3.1.4.1.

2.2.1.4 PTUNNEL_CONTEXT_HANDLE_SERIALIZE

An RPC context handle representing the tunnel for the given connection. Refer to [MSDN-MMSCH] for details on the modes of the context handles. For this context handle, there can be no more than one pending RPC call on the TSG server. On the wire it is identical to PTUNNEL_CONTEXT_HANDLE_NOSERIALIZE.

This type is declared as follows:

```
typedef [context_handle] PTUNNEL_CONTEXT_HANDLE_NOSERIALIZE PTUNNEL_CONTEXT_HANDLE_SERIALIZE;
```

The context handle MUST NOT be type_strict, but it MUST be strict. More details on RPC context handles are specified in [C706] sections 4.2.16.6, 5.1.6, and 6.1 and [MS-RPCE] section 3.1.1.5.3.2.2.2 and 3.3.1.4.1.

2.2.1.5 PCHANNEL_CONTEXT_HANDLE_SERIALIZE

An RPC context handle representing the channel for the given connection. Refer to [MSDN-MMSCH] for details on the modes of the context handles. For this context handle, there can be no more than one pending RPC call on the TSG server. On the wire it is identical to PCHANNEL_CONTEXT_HANDLE_NOSERIALIZE.

This type is declared as follows:

```
typedef [context_handle] PCHANNEL_CONTEXT_HANDLE_NOSERIALIZE
PCHANNEL_CONTEXT_HANDLE_SERIALIZE;
```

The context handle MUST NOT be type_strict, but it MUST be strict. More details on RPC context handles are specified in [C706] sections 4.2.16.6, 5.1.6, and 6.1 and [MS-RPCE] section 3.1.1.5.3.2.2.2 and 3.3.1.4.1.

2.2.2 Constants

2.2.2.1 MAX_RESOURCE_NAMES

Constant/value	Description
MAX_RESOURCE_NAMES 50	The maximum range allowed by the TSG server for the **numResourceNames** data type in the **TSENDPOINTINFO** structure.

2.2.2.2 TSG_PACKET_TYPE_HEADER

Constant/value	Description
TSG_PACKET_TYPE_HEADER 0x00004844	This constant is used by the **packetId** field of the **TSG_PACKET** structure. The TSG client and TSG server SHOULD not use this type, as specified in sections 2.2.3.2 and 2.2.3.2.1.1.

2.2.2.3 TSG_PACKET_TYPE_VERSIONCAPS

Constant/value	Description
TSG_PACKET_TYPE_VERSIONCAPS 0x00005643	This constant is used by the **packetId** field of the **TSG_PACKET** structure. When this constant is present, the **packetVersionCaps** field of the **tsgPacket** union field in the **TSG_PACKET** structure MUST be a pointer to a **TSG_PACKET_VERSIONCAPS** structure.

2.2.2.4 TSG_PACKET_TYPE_QUARCONFIGREQUEST

Constant/value	Description
TSG_PACKET_TYPE_QUARCONFIGREQUEST 0x00005143	This constant is used by the **packetId** field of the **TSG_PACKET** structure. When this constant is present, the **packetQuarConfigRequest** field of the **tsgPacket** union field in the **TSG_PACKET** structure MUST be a pointer to a **TSG_PACKET_QUARCONFIGREQUEST** structure.

2.2.2.5 TSG_PACKET_TYPE_QUARREQUEST

Constant/value	Description
TSG_PACKET_TYPE_QUARREQUEST 0x00005152	This constant is used by the **packetId** field of the TSG_PACKET structure. When this constant is present, the **packetQuarRequest** field of the **tsgPacket** union field in the **TSG_PACKET** structure MUST be a pointer to a TSG_PACKET_QUARREQUEST structure. It is also used by the TSG server in the flags filed of the TSG_PACKET_RESPONSE structure in response to the TsProxyAuthorizeTunnel call.

2.2.2.6 TSG_PACKET_TYPE_RESPONSE

Constant/value	Description
TSG_PACKET_TYPE_RESPONSE 0x00005052	This constant is used by the **packetId** field, of the TSG_PACKET structure. When this constant is present, the **packetResponse** field of the **tsgPacket** union field, in the **TSG_PACKET** structure, MUST be a pointer to a TSG_PACKET_RESPONSE structure.

2.2.2.7 TSG_PACKET_TYPE_QUARENC_RESPONSE

Constant/value	Description
TSG_PACKET_TYPE_QUARENC_RESPONSE 0x00004552	This constant is used by the **packetId** field of the TSG_PACKET structure. When this type is present, the **packetQuarEncResponse** field of the **tsgPacket** union field in the **TSG_PACKET** structure MUST be a pointer to a TSG_PACKET_QUARENC_RESPONSE structure.

2.2.2.8 TSG_CAPABILITY_TYPE_NAP

Constant/value	Description
TSG_CAPABILITY_TYPE_NAP 0x1	This constant is used by the **tsgCapNap** field of TSG_CAPABILITIES_UNION. It indicates whether **Network Access Protection (NAP)** capabilities are supported by the TSG client and TSG server.

2.2.2.9 TSG_PACKET_TYPE_CAPS_RESPONSE

Constant/value	Description
TSG_PACKET_TYPE_CAPS_RESPONSE 0x00004350	This constant is used by the **packetId** field of the TSG_PACKET structure. When this type is present, the **packetCapsResponse** field of the **tsgPacket** union field in the **TSG_PACKET** structure MUST be a pointer to a TSG_PACKET_CAPS_RESPONSE structure.

2.2.2.10 TSG_PACKET_TYPE_MSGREQUEST_PACKET

Constant/value	Description
TSG_PACKET_TYPE_MSGREQUEST_PACKET	This constant is used by the **packetId** field of the TSG_PACKET structure. When this type is present, the

Constant/value	Description
0x00004752	**packetMsgRequest** field of the **tsgPacket** union field in the **TSG_PACKET** structure MUST be a pointer to a TSG_PACKET_MSG_REQUEST structure.

2.2.2.11 TSG_PACKET_TYPE_MESSAGE_PACKET

Constant/value	Description
TSG_PACKET_TYPE_MESSAGE_PACKET 0x00004750	This constant is used by the **packetId** field of the **TSG_PACKET** structure. When this type is present, the **packetMsgResponse** field of the **tsgPacket** union field in the **TSG_PACKET** structure MUST be a pointer to a TSG_PACKET_MSG_RESPONSE structure.

2.2.2.12 TSG_PACKET_TYPE_AUTH

Constant/value	Description
TSG_PACKET_TYPE_AUTH 0x00004054	This constant is used by the **packetId** field of the **TSG_PACKET** structure. When this type is present, the **packetAuth** field of the **tsgPacket** union field in the **TSG_PACKET** structure MUST be a pointer to a TSG_PACKET_AUTH structure.

2.2.2.13 TSG_PACKET_TYPE_REAUTH

Constant/value	Description
TSG_PACKET_TYPE_REAUTH 0x00005250	This constant is used by the **packetId** field of the **TSG_PACKET** structure. When this type is present, the **packetReauth** field of the **tsgPacket** union field in the **TSG_PACKET** structure MUST be a pointer to a TSG_PACKET_REAUTH structure.

2.2.2.14 TSG_ASYNC_MESSAGE_CONSENT_MESSAGE

Constant/value	Description
TSG_ASYNC_MESSAGE_CONSENT_MESSAGE 0x00000001	This constant is used by the **msgType** field of the TSG_PACKET_MSG_RESPONSE structure. This value indicates that the **consentMessage** field of the TSG_PACKET_TYPE_MESSAGE_UNION contains the Consent Message.

2.2.2.15 TSG_ASYNC_MESSAGE_SERVICE_MESSAGE

Constant/value	Description
TSG_ASYNC_MESSAGE_SERVICE_MESSAGE 0x00000002	This constant is used by the **msgType** field of the TSG_PACKET_MSG_RESPONSE structure. This value indicates that the **serviceMessage** field of the TSG_PACKET_TYPE_MESSAGE_UNION contains the Service Message.

2.2.2.16 TSG_ASYNC_MESSAGE_REAUTH

Constant/value	Description
TSG_ASYNC_MESSAGE_REAUTH 0x00000003	This constant is used by the **msgType** field of the TSG_PACKET_MSG_RESPONSE structure. This value indicates that the **reauthMessage** field of the TSG_PACKET_TYPE_MESSAGE_UNION contains the Re-authentication request to the client.

2.2.2.17 TSG_TUNNEL_CALL_ASYNC_MSG_REQUEST

Constant/value	Description
TSG_TUNNEL_CALL_ASYNC_MSG_REQUEST 0x00000001	This constant is used by the *procId* parameter of the TsProxyMakeTunnelCall method. This value indicates that the client can receive Service Messages and the TSG server SHOULD send the same when available.

2.2.2.18 TSG_TUNNEL_CANCEL_ASYNC_MSG_REQUEST

Constant/value	Description
TSG_TUNNEL_CANCEL_ASYNC_MSG_REQUEST 0x00000002	This constant is used by the *procId* parameter of the **TsProxyMakeTunnelCall** method. This value indicates that the client has requested to cancel the pending service message request on the TSG server.

2.2.2.19 TSG_NAP_CAPABILITY_QUAR_SOH

Constant/value	Description
TSG_NAP_CAPABILITY_QUAR_SOH 0x00000001	This constant is used to represent the NAP quarantine **statement of health (SoH)** capability. If the TSG client supports this capability, it means that the TSG client is capable of sending a quarantine **statement of health response (SoHR)** to the TSG server as specified in section 2.2.3.2.1.5.1. If the TSG server supports this capability, it means that the TSG server is capable of receiving and processing a quarantine statement of health response from the TSG client as specified in section 2.2.3.2.1.5.1.<2>

2.2.2.20 TSG_NAP_CAPABILITY_IDLE_TIMEOUT

Constant/value	Description
TSG_NAP_CAPABILITY_IDLE_TIMEOUT 0x00000002	This constant is used to represent the Idle timeout capability. If the TSG client supports this capability, it means that the TSG client is capable of receiving and processing an idle timeout value as specified in section 2.2.3.2.1.5.1. If the TSG server supports this capability, it means that the TSG server is capable of sending an idle timeout value to the client as specified in section 2.2.3.2.1.5.1.

2.2.2.21 TSG_MESSAGING_CAP_CONSENT_SIGN

Constant/value	Description
TSG_MESSAGING_CAP_CONSENT_SIGN 0x00000004	This constant is used to represent the consent message capability. If the TSG client supports this capability, it means that the TSG client is capable of receiving and processing a consent message as specified in section 2.2.3.2.1.9.1. If the TSG server supports this capability, it means that the TSG server is capable of sending a consent message to the TSG client as specified in section 2.2.3.2.1.9.1.

2.2.2.22 TSG_MESSAGING_CAP_SERVICE_MSG

Constant/value	Description
TSG_MESSAGING_CAP_SERVICE_MSG 0x00000008	This constant is used to represent the service message capability. If the TSG client supports this capability, it means that the TSG client is capable of receiving and processing a service message as specified in section 2.2.3.2.1.9.1. If the TSG server supports this capability, it means that the TSG server is capable of sending a service message to the TSG client as specified in section 2.2.3.2.1.9.1.

2.2.2.23 TSG_MESSAGING_CAP_REAUTH

Constant/value	Description
TSG_MESSAGING_CAP_REAUTH 0x00000010	This constant is used to represent the **re-authentication** capability. If the TSG client supports this capability, it means that the TSG client is capable of performing re-authentication according to the same methods as initial authentication, as specified in section 2.1. If the TSG server supports this capability, it means that the TSG server is capable of sending a re-authentication request to the TSG client, as specified in section 2.2.3.2.1.9.1

2.2.2.24 Return Codes

The following **HRESULT** return values are specified by this protocol. The protocol must be ended when any of the below return codes, except ERROR_SUCCESS, are received. The phrase "ending the protocol" refers to closing the channel and tunnel, if a channel has been created; or closing the tunnel, if a channel has not been created, but the tunnel has been created. To close the channel, **TsProxyCloseChannel (section 3.1.4.3.1)** must be called, and to close the tunnel, **TsProxyCloseTunnel (section 3.1.4.3.3)** must be called. When these calls are completed, the protocol ends. After the protocol ends, the binding handle should be closed. A binding handle is specified in [C706] section 2.<3>

Return value/code	Description
0x800759D8 E_PROXY_INTERNALERROR	Used as a generic catch-all when an unexpected error happens.
0x800759DA E_PROXY_RAP_ACCESSDENIED	Returned when an attempt to resolve or access a target server is blocked by TSG server policies.

Return value/code	Description
0x800759DB E_PROXY_NAP_ACCESSDENIED	Returned when the TSG server denies the TSG client access due to policy.
0x800759DF E_PROXY_ALREADYDISCONNECTED	Returned when an operation is called on a disconnected tunnel or channel.
0x800759ED E_PROXY_QUARANTINE_ACCESSDENIED	The TSG server rejects the connection due to quarantine policy.
0x800759EE E_PROXY_NOCERTAVAILABLE	The TSG server cannot find a certificate to register for SCHANNEL **Authentication Service (AS)**.
0x800759F7 E_PROXY_COOKIE_BADPACKET	An invalid cookie packet was sent by the client.
0x800759F8 E_PROXY_COOKIE_AUTHENTICATION_ACCESS_DENIED	Returned when the TSG server is in pluggable authentication mode and the given user does not have access to connect via TSG server.
0x800759F9 E_PROXY_UNSUPPORTED_AUTHENTICATION_METHOD	Returned to the TSG client from the call to TsProxyCreateTunnel when the TSG server is configured for pluggable authentication and the value of the **packetId** member of the *tsgPacket* parameter is not equal to TSG_PACKET_TYPE_AUTH or TSG_PACKET_TYPE_REAUTH.
0x800759E9 E_PROXY_CAPABILITYMISMATCH	Returned when the TSG server supports the TSG_MESSAGING_CAP_CONSENT_SIGN capability and is configured to allow only a TSG client that supports the **TSG_MESSAGING_CAP_CONSENT_SIGN** capability, but the TSG client doesn't support the capability.
0x8007071A HRESULT_FROM_WIN32(RPC_S_CALL_CANCELLED)	Returned when a pending call is canceled by the TSG client or the call is canceled because a shutdown sequence is initiated.
0x00000000 ERROR_SUCCESS	Returned when the requested operation succeeds.

In addition to the preceding HRESULTs, which are defined by the [MS-TSGU] protocol, the following **DWORDs** are returned in an rpc_fault packet when an exception is raised on the TSG server.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	Returned by the TSG server when the requested operation is not allowed.
0x000004E3 ERROR_ONLY_IF_CONNECTED	Returned by the TSG server when an attempt is made by the client to send data to the target server on connection state other than Pipe Created state.

Return value/code	Description
0x00000057 ERROR_INVALID_PARAMETER	Returned by the TSG server when the TSG client sends a non-NULL value in a data member of the **TSG_PACKET_QUARREQUEST** structure but it is not prefixed with Nonce.
0x000004CA ERROR_GRACEFUL_DISCONNECT	Returned by the TSG server when the connection is disconnected gracefully by the TSG client by calling **TsProxyCloseChannel**.
0x000059E8 HRESULT_CODE(E_PROXY_NOTSUPPORTED)	Returned when the TSG server receives a wrong packet in the **TsProxyAuthorizeTunnel** method.
0x000003E3 ERROR_OPERATION_ABORTED	Returned when the TSG server does not receive a **TsProxySetupReceivePipe** method call before the Connection Timer (section 3.1.2.3) expires.
0x000059DD HRESULT_CODE(E_PROXY_TS_CONNECTFAILED)	Returned by **TsProxyCreateChannel** when the TSG server fails to connect to the target server.<4>
0x000059E6 HRESULT_CODE(E_PROXY_MAXCONNECTIONSREACHED)	The TSG server has reached the maximum connections allowed.<5>

In addition to the above **DWORDs**, the following **DWORDs** are returned from the **TsProxySetupReceivePipe** and **TsProxySendToServer** methods.

Return value/code	Description
0X000059D8 HRESULT_CODE(E_PROXY_INTERNALERROR)	Returned when an unexpected error occurs in **TsProxySetupReceivePipe** or **TsProxySendToServer**.
0x000059DD HRESULT_CODE(E_PROXY_TS_CONNECTFAILED)	Returned by **TsProxySetupReceivePipe** when the TSG server fails to connect to the target server. It is returned in the rpc_fault packet.<6>
0x000059F6 HRESULT_CODE(E_PROXY_SESSIONTIMEOUT)	Returned by **TsProxySetupReceivePipe** if a session timeout occurs and "disconnect on session timeout" is configured at the TSG server and the ADM element **Negotiated Capabilities** contains **TSG_NAP_CAPABILITY_IDLE_TIMEOUT**.
0X000059FA HRESULT_CODE(E_PROXY_REAUTH_AUTHN_FAILED)	Returned by **TsProxySetupReceivePipe** when a re-authentication attempt by the client has failed because the user credentials are no longer valid and the ADM element **Negotiated Capabilities** contains **TSG_NAP_CAPABILITY_IDLE_TIMEOUT**.
0x000059FB HRESULT_CODE(E_PROXY_REAUTH_CAP_FAILED)	Returned by **TsProxySetupReceivePipe** when a re-authentication attempt by the client has

Return value/code	Description
	failed because the user is not authorized to connect through the TSG server anymore and the ADM element **Negotiated Capabilities** contains **TSG_NAP_CAPABILITY_IDLE_TIMEOUT**.
0x000059FC HRESULT_CODE(E_PROXY_REAUTH_RAP_FAILED)	Returned by **TsProxySetupReceivePipe** when a re-authentication attempt by the client has failed because the user is not authorized to connect to the given end resource anymore and the ADM element **Negotiated Capabilities** contains **TSG_NAP_CAPABILITY_IDLE_TIMEOUT**.
0x000059FD HRESULT_CODE(E_PROXY_SDR_NOT_SUPPORTED_BY_TS)	The TSG server is capable of exchanging policies with some target servers.<7> If the TSG server is configured to allow connections to only target servers that are capable of policy exchange and the target server is not capable of exchanging policies with the TSG server, this error will be returned by **TsProxySetupReceivePipe**.
0x00005A00 HRESULT_CODE(E_PROXY_REAUTH_NAP_FAILED)	Returned by **TsProxySetupReceivePipe** when a re-authentication attempt by the TSG client has failed because the health of the user's computer is no longer compliant with the TSG server configuration and the ADM element **Negotiated Capabilities** contains **TSG_NAP_CAPABILITY_IDLE_TIMEOUT**.
0x000004D4 HRESULT_CODE(E_PROXY_CONNECTIONABORTED)	Returned by **TsProxySetupReceivePipe** when the following happens: 1. The TSG server administrator forcefully disconnects the connection. 2. Or when the ADM element **Negotiated Capabilities** doesn't contain **TSG_NAP_CAPABILITY_IDLE_TIMEOUT** and any one of the following happens: 1. Session timeout occurs and disconnect on session timeout is configured at the TSG server. 2. Re-authentication attempt by the client has failed because the user credentials are no longer valid. 3. Re-authentication attempt by the client has failed because the user is not authorized to connect through the TSG server anymore. 4. Re-authentication attempt by the

Return value/code	Description
	client has failed because the user is not authorized to connect to the given end resource anymore. 5. Re-authentication attempt by the TSG client has failed because the health of the user's computer is no longer compliant with the TSG server configuration.
0x000000A0 ERROR_BAD_ARGUMENTS	Returned by **TsProxySetupReceivePipe** when the target server unexpectedly closes the connection between the TSG server and the target server.

2.2.3 Structures and Unions

2.2.3.1 TSENDPOINTINFO

The **TSENDPOINTINFO** structure contains information about the target server to which the TSG server attempts to connect.

```
typedef struct _tsendpointinfo {
  [size_is(numResourceNames)] RESOURCENAME* resourceName;
  [range(0, MAX_RESOURCE_NAMES)] unsigned long numResourceNames;
  [unique, size_is(numAlternateResourceNames)]
    RESOURCENAME* alternateResourceNames;
  [range(0, 3)] unsigned short numAlternateResourceNames;
  unsigned long Port;
} TSENDPOINTINFO,
 *PTSENDPOINTINFO;
```

resourceName: An array of **RESOURCENAME** strings, as specified in section 2.2.1.1. The range is from 0 to **numResourceNames**. This array, in conjunction with *alternateResourceNames* parameter array, comprises the alias names of the target server to which the TSG server can connect. As specified in the Protocol Overview (section 1.3), the TSG server acts as a proxy to target server. The RDP client and target server MUST use [MS-RDPBCGR] to communicate.

numResourceNames: The number of **RESOURCENAME** datatypes in the **resourceName** array. The value must be in the range of 0 and 3; both inclusive.

alternateResourceNames: An array of **RESOURCENAME** strings to be used as alternative names for the target server. The range is from 0 to **numAlternateResourceNames**.<8>

numAlternateResourceNames: The number of allowed **alternateResourceNames**. The value must be in the range of 0 and 3; both inclusive.

Port: Specifies the protocol ID and TCP port number for the target server endpoint to which the TSG server connects. The protocol ID is in the low order 16 bits of this field and port number is in the high order 16 bits. These values can be ignored. The value of the protocol id is protocol-dependent. For example, RDP uses 3.

2.2.3.2 TSG_PACKET

The **TSG_PACKET** structure specifies the type of structure to be used by the TSG client and TSG server.

```
typedef struct _TSG_PACKET {
  unsigned long packetId;
  [switch_is(packetId)] TSG_PACKET_TYPE_UNION tsgPacket;
} TSG_PACKET,
 *PTSG_PACKET;
```

packetId: This value specifies the type of structure pointer contained in the **tsgPacket** field. Valid values are specified in sections 2.2.2.2, 2.2.2.3, 2.2.2.4, 2.2.2.5, 2.2.2.6, 2.2.2.7, 2.2.2.9, 2.2.2.10, 2.2.2.11, 2.2.2.12, and 2.2.2.13.

tsgPacket: A union field containing the actual structure pointer corresponding to the value contained in the **packetId** field. Valid structures for this field are specified in sections 2.2.3.2.1.1, 2.2.3.2.1.2, 2.2.3.2.1.3, 2.2.3.2.1.4, 2.2.3.2.1.5, 2.2.3.2.1.6, 2.2.3.2.1.7, 2.2.3.2.1.8, 2.2.3.2.1.9, 2.2.3.2.1.10, and 2.2.3.2.1.11.

2.2.3.2.1 TSG_PACKET_TYPE_UNION

The **TSG_PACKET_TYPE_UNION** union specifies an RPC switch_type union of structures as follows.

```
typedef
[switch_type(unsigned long)]
  union {
  [case(TSG_PACKET_TYPE_HEADER)]
    PTSG_PACKET_HEADER packetHeader;
  [case(TSG_PACKET_TYPE_VERSIONCAPS)]
    PTSG_PACKET_VERSIONCAPS packetVersionCaps;
  [case(TSG_PACKET_TYPE_QUARCONFIGREQUEST)]
    PTSG_PACKET_QUARCONFIGREQUEST packetQuarConfigRequest;
  [case(TSG_PACKET_TYPE_QUARREQUEST)]
    PTSG_PACKET_QUARREQUEST packetQuarRequest;
  [case(TSG_PACKET_TYPE_RESPONSE)]
    PTSG_PACKET_RESPONSE packetResponse;
  [case(TSG_PACKET_TYPE_QUARENC_RESPONSE)]
    PTSG_PACKET_QUARENC_RESPONSE packetQuarEncResponse;
  [case(TSG_PACKET_TYPE_CAPS_RESPONSE)]
    PTSG_PACKET_CAPS_RESPONSE packetCapsResponse;
  [case(TSG_PACKET_TYPE_MSGREQUEST_PACKET)]
    PTSG_PACKET_MSG_REQUEST packetMsgRequest;
  [case(TSG_PACKET_TYPE_MESSAGE_PACKET)]
    PTSG_PACKET_MSG_RESPONSE packetMsgResponse;
  [case(TSG_PACKET_TYPE_AUTH)]
    PTSG_PACKET_AUTH packetAuth;
  [case(TSG_PACKET_TYPE_REAUTH)]
    PTSG_PACKET_REAUTH packetReauth;
} TSG_PACKET_TYPE_UNION,
 *PTSG_PACKET_TYPE_UNION;
```

packetHeader: A **PTSG_PACKET_HEADER** as specified in section 2.2.3.2.1.1.

packetVersionCaps: A **PTSG_PACKET_VERSIONCAPS** as specified in section 2.2.3.2.1.2.

packetQuarConfigRequest: A **PTSG_PACKET_QUARCONFIGREQUEST** as specified in section 2.2.3.2.1.3.

packetQuarRequest: A **PTSG_PACKET_QUARREQUEST** as specified in section 2.2.3.2.1.4.

packetResponse: A **PTSG_PACKET_RESPONSE** as specified in section 2.2.3.2.1.5.

packetQuarEncResponse: A **PTSG_PACKET_QUARENC_RESPONSE** as specified in section 2.2.3.2.1.6.

packetCapsResponse: A **PTSG_PACKET_CAPS_RESPONSE** as specified in section 2.2.3.2.1.7.

packetMsgRequest: A **PTSG_PACKET_MSG_REQUEST** as specified in section 2.2.3.2.1.8.

packetMsgResponse: A **PTSG_PACKET_MSG_RESPONSE** as specified in section 2.2.3.2.1.9.

packetAuth: A **PTSG_PACKET_AUTH** as specified in section 2.2.3.2.1.10.

packetReauth: A **PTSG_PACKET_REAUTH** as specified in section 2.2.3.2.1.11.

2.2.3.2.1.1 TSG_PACKET_HEADER

The **TSG_PACKET_HEADER** structure contains information about the **ComponentID** and **PacketID** fields of the **TSG_PACKET** structure. The value of **PacketID** in **TSG_PACKET** MUST be set to **TSG_PACKET_TYPE_HEADER**.

```
typedef struct _TSG_PACKET_HEADER {
  unsigned short ComponentId;
  unsigned short PacketId;
} TSG_PACKET_HEADER,
 *PTSG_PACKET_HEADER;
```

ComponentId: Represents the component sending the packet. This MUST be the following value:

Value	Meaning
0x5452	TS Gateway Transport

PacketId: Unused.

This structure cannot be used by itself as part of any method call. It can be used only in the context of other structures.

2.2.3.2.1.2 TSG_PACKET_VERSIONCAPS

The **TSG_PACKET_VERSIONCAPS** structure is used for version and capabilities negotiation. The value of the **packetId** field in TSG_PACKET must be set to TSG_PACKET_TYPE_VERSIONCAPS.

This structure must be embedded in the TSG_PACKET_QUARENC_RESPONSE.

```
typedef struct _TSG_PACKET_VERSIONCAPS {
  TSG_PACKET_HEADER tsgHeader;
  [size_is(numCapabilities)] PTSG_PACKET_CAPABILITIES tsgCaps;
  [range(0, 32)] unsigned long numCapabilities;
  unsigned short majorVersion;
  unsigned short minorVersion;
  unsigned short quarantineCapabilities;
} TSG_PACKET_VERSIONCAPS,
 *PTSG_PACKET_VERSIONCAPS;
```

tsgHeader: Specified in 2.2.3.2.1.1.

tsgCaps: An array of TSG_PACKET_CAPABILITIES structures. The number of elements in the array is indicated by the **numCapabilities** field.

numCapabilities: The number of array elements for the **tsgCaps** field. This value must be in the range of 0 and 32. If the **tsgCaps** field is ignored, then this field must also be ignored.

majorVersion: Indicates the major version of the TSG client or TSG server, depending on the sender. This MUST be the following value:

Value	Meaning
0x0001	Current major version of the Terminal Services Gateway Server Protocol.

minorVersion: Indicates the minor version of the TSG client or TSG server, depending on the sender. This MUST be the following value.

Value	Meaning
0x0001	Current minor version of the Terminal Services Gateway Server Protocol.

quarantineCapabilities: Indicates quarantine capabilities of the TSG client and TSG server, depending on the sender. This MAY be the following value:<9>

Value	Meaning
0x0001	Quarantine is supported and required by the TSG server.

2.2.3.2.1.2.1 TSG_PACKET_CAPABILITIES

The **TSG_PACKET_CAPABILITIES** structure contains information about the capabilities of the TSG client and TSG server.

This structure must be embedded in the TSG_PACKET_VERSIONCAPS structure.

```
typedef struct _TSG_PACKET_CAPABILITIES {
  unsigned long capabilityType;
  [switch_is(capabilityType)] TSG_CAPABILITIES_UNION tsgPacket;
} TSG_PACKET_CAPABILITIES,
 *PTSG_PACKET_CAPABILITIES;
```

capabilityType: Indicates the type of NAP capability supported by the TSG client or the TSG server. This member MUST be the following value:

Value	Meaning
0x00000001	The TSG server supports NAP capability type (TSG_CAPABILITY_TYPE_NAP).<10>

tsgPacket: Specifies the union containing the actual structure corresponding to the value defined in the **capabilityType** field. Valid structures are specified in sections 2.2.3.2.1.2.1.1 and 2.2.3.2.1.2.1.2.

2.2.3.2.1.2.1.1 TSG_CAPABILITIES_UNION

The **TSG_CAPABILITIES_UNION** union specifies an RPC switch_type union of structures as follows.

```
typedef
[switch_type(unsigned long)]
  union {
  [case(TSG_CAPABILITY_TYPE_NAP)]
    TSG_CAPABILITY_NAP tsgCapNap;
} TSG_CAPABILITIES_UNION,
 *PTSG_CAPABILITIES_UNION;
```

tsgCapNap: A **TSG_CAPABILITY_NAP** structure.

2.2.3.2.1.2.1.2 TSG_CAPABILITY_NAP

The **TSG_CAPABILITY_NAP** structure contains information about the NAP capabilities of the TSG client and TSG server.

This structure must be embedded in the **TSG_PACKET_CAPABILITIES** structure.

```
typedef struct _TSG_CAPABILITY_NAP {
  unsigned long capabilities;
} TSG_CAPABILITY_NAP,
 *PTSG_CAPABILITY_NAP;
```

capabilities: Indicates the NAP capabilities supported by the TSG client and TSG server. This bit field MUST be 0 or one or more of the following values.

Value
TSG_NAP_CAPABILITY_QUAR_SOH

Value
TSG_NAP_CAPABILITY_IDLE_TIMEOUT
TSG_MESSAGING_CAP_CONSENT_SIGN
TSG_MESSAGING_CAP_SERVICE_MSG
TSG_MESSAGING_CAP_REAUTH

2.2.3.2.1.3 TSG_PACKET_QUARCONFIGREQUEST

The TSG_PACKET_QUARCONFIGREQUEST structure contains information about quarantine configuration. TSG server and TSG client MAY support this structure.<11> If the TSG server or TSG client do not support the TSG_PACKET_QUARCONFIGREQUEST structure, then the error code HRESULT_CODE(E_PROXY_NOTSUPPORTED) is returned.

```
typedef struct _TSG_PACKET_QUARCONFIGREQUEST {
  unsigned long flags;
} TSG_PACKET_QUARCONFIGREQUEST,
 *PTSG_PACKET_QUARCONFIGREQUEST;
```

flags: Contains information about quarantine configuration.

2.2.3.2.1.4 TSG_PACKET_QUARREQUEST

The **TSG_PACKET_QUARREQUEST** structure<12> contains information about the TSG client's statement of health (SoH) and the name of the TSG client machine. The value of the **packetId** field in **TSG_PACKET** MUST be set to TSG_PACKET_TYPE_QUARREQUEST.

```
typedef struct _TSG_PACKET_QUARREQUEST {
  unsigned long flags;
  [string, size_is(nameLength)] wchar_t* machineName;
  [range(0, 512 + 1)] unsigned long nameLength;
  [unique, size_is(dataLen)] byte* data;
  [range(0, 8000)] unsigned long dataLen;
} TSG_PACKET_QUARREQUEST,
 *PTSG_PACKET_QUARREQUEST;
```

flags: This field can be any value when sending and ignored on receipt.

machineName: A string representing the name of the TSG Client Machine Name (section 3.2.1).<13> This field can be ignored. The length of the name, including the terminating null character, MUST be equal to the size specified by the **nameLength** field.

nameLength: An unsigned long specifying the number of characters in **machineName**, including the terminating null character. The specified value MUST be in the range from 0 to 513 characters.

data: An array of bytes that specifies the statement of health prepended with nonce, which is obtained in **TSG_PACKET_QUARENC_RESPONSE (section 2.2.3.2.1.6)** from the TSG server in response to **TsProxyCreateTunnel**.<14> This field can be ignored. The length of this data is specified by the **dataLen** field.

dataLen: The length, in bytes, of the **data** field. This value MUST be in the range between 0 and 8000, both inclusive.

2.2.3.2.1.5 TSG_PACKET_RESPONSE

The **TSG_PACKET_RESPONSE** structure contains the response of the TSG server to the TSG client for the **TsProxyAuthorizeTunnel** method call. The value of the **packetId** field in **TSG_PACKET** MUST be set to **TSG_PACKET_TYPE_RESPONSE**.

```
typedef struct _TSG_PACKET_RESPONSE {
  unsigned long flags;
  unsigned long reserved;
  [size_is(responseDataLen)] byte* responseData;
  [range(0, 24000)] unsigned long responseDataLen;
  TSG_REDIRECTION_FLAGS redirectionFlags;
} TSG_PACKET_RESPONSE,
 *PTSG_PACKET_RESPONSE;
```

flags: The TSG server MUST set this value to **TSG_PACKET_TYPE_QUARREQUEST** to indicate that this structure is in response to the **TsProxyAuthorizeTunnel** method call. The TSG client MAY ignore this field.

reserved: This field is unused and can be any value when sending and ignored on receipt.

responseData: Byte data representing the response from the TSG server for the **TsProxyAuthorizeTunnel** method call. If the ADM element **Negotiated Capabilities** contains TSG_NAP_CAPABILITY_QUAR_SOH and TSG_NAP_CAPABILITY_IDLE_TIMEOUT, then **responseData** MUST contain both the statement of health response (SoHR) and the idle timeout value. If **Negotiated Capabilities** contains only TSG_NAP_CAPABILITY_QUAR_SOH, then **responseData** MUST contain only the statement of health response. If **Negotiated Capabilities** contains only TSG_NAP_CAPABILITY_IDLE_TIMEOUT, then **responseData** MUST contain only the idle timeout value. The length of the data MUST be equal to that specified by **responseDataLen**. If **Negotiated Capabilities** does not contain both TSG_NAP_CAPABILITY_QUAR_SOH and TSG_NAP_CAPABILITY_IDLE_TIMEOUT, then **responseData** is ignored and **responseDataLen** is set to zero.<15>

responseDataLen: Length, in bytes, of the data specified by the **responseData** field.

redirectionFlags: A **TSG_REDIRECTION_FLAGS** structure.<16>

2.2.3.2.1.5.1 responseData Format

The TSG server uses the responseData to send various data to the TSG client after tunnel authorization. The responseData is shown below.

Note Both the **Idle timeout value** and **Statement of health response** fields are optional, meaning either one of them or both can be absent. Also note that, in case of **Idle timeout value** absence, **Statement of health response** begins from the first **DWORD** itself. If both of them are absent, the responseData is ignored and **responseDataLen** is set to zero.

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
Idle timeout value (optional)																															
Statement of health response (variable)																															
...																															

Idle timeout value (4 bytes): If the ADM element **Negotiated Capabilities** contains **TSG_NAP_CAPABILITY_IDLE_TIMEOUT**, then the first 4 bytes of the **responseData** field is the **Idle timeout value** in units of minutes.

Statement of health response (variable): If the ADM element **Negotiated Capabilities** contains **TSG_NAP_CAPABILITY_QUAR_SOH** and the **Statement of health** is passed in the **TsProxyAuthorizeTunnel** call as specified in 2.2.3.2.1.4, then the remaining number of bytes of the **responseData** field is the **Statement of health** response.

2.2.3.2.1.5.2 TSG_REDIRECTION_FLAGS

The **TSG_REDIRECTION_FLAGS** structure specifies the device redirection settings that MUST be enforced by the TSG client. For details about device redirection, see [MS-TSSO] section 3.1.6.

This structure MUST be embedded in the **TSG_PACKET_RESPONSE** structure.

Note Both **enableAllRedirections** and **disableAllRedirections** must not be TRUE.

```
typedef struct _TSG_REDIRECTION_FLAGS {
  BOOL enableAllRedirections;
  BOOL disableAllRedirections;
  BOOL driveRedirectionDisabled;
  BOOL printerRedirectionDisabled;
  BOOL portRedirectionDisabled;
  BOOL reserved;
  BOOL clipboardRedirectionDisabled;
  BOOL pnpRedirectionDisabled;
} TSG_REDIRECTION_FLAGS,
 *PTSG_REDIRECTION_FLAGS;
```

enableAllRedirections: A Boolean value indicating whether the TSG server specifies any control over the device redirection on the TSG client.

Value	Meaning
FALSE 0x00000000	Device redirection is not enabled for all devices. Other fields of this structure specify which device redirection is enabled or disabled.
TRUE 0x00000001	Device redirection is enabled for all devices. All other fields of this structure must be ignored.

disableAllRedirections: A Boolean value indicating whether the TSG server specifies any control over disabling all device redirection on the TSG client.

Value	Meaning
FALSE 0x00000000	Device redirection is not disabled for all devices. Other fields of this structure specify which device redirection is enabled or disabled.
TRUE 0x00000001	Device redirection is disabled for all devices. All other fields of this structure must be ignored.

driveRedirectionDisabled: A Boolean value indicating whether the TSG server specifies any control over disabling drive redirection on the TSG client.

Value	Meaning
FALSE 0x00000000	The TSG client is allowed to choose its own redirection settings for enabling or disabling drive redirection.
TRUE 0x00000001	Drive redirection is disabled.

printerRedirectionDisabled: A Boolean value indicating whether the TSG server specifies any control over disabling printer redirection on the TSG client.

Value	Meaning
FALSE 0x00000000	The TSG client is allowed to choose its own redirection settings for enabling or disabling printer redirection.
TRUE 0x00000001	Printer redirection is disabled.

portRedirectionDisabled: A Boolean value indicating whether the TSG server specifies any control over disabling port redirection on the TSG client.

Value	Meaning
FALSE 0x00000000	The TSG client is allowed to choose its own redirection settings for enabling or disabling port redirection. Port redirection applies to both serial (COM) and parallel ports (LPT).
TRUE 0x00000001	Port redirection is disabled.

reserved: Unused. MUST be 0.

clipboardRedirectionDisabled: A Boolean value indicating whether the TSG server specifies any control over disabling clipboard redirection on the TSG client.

Value	Meaning
FALSE 0x00000000	The TSG client is allowed to choose its own redirection settings for enabling or disabling clipboard redirection.
TRUE 0x00000001	Clipboard redirection is disabled.

pnpRedirectionDisabled: A Boolean value indicating whether the TSG server specifies any control over disabling Plug and Play redirection on the TSG client.

Value	Meaning
FALSE 0x00000000	The TSG client is allowed to choose its own redirection settings for enabling or disabling PnP redirection.
TRUE 0x00000001	PnP redirection is disabled.

2.2.3.2.1.6 TSG_PACKET_QUARENC_RESPONSE

The **TSG_PACKET_QUARENC_RESPONSE** structure contains the response of the TSG server for the **TsProxyCreateTunnel** method call. The value of the **packetId** field in **TSG_PACKET** MUST be set to **TSG_PACKET_TYPE_QUARENC_RESPONSE**.

```
typedef struct _TSG_PACKET_QUARENC_RESPONSE {
  unsigned long flags;
  [range(0, 24000)] unsigned long certChainLen;
  [string, size_is(certChainLen)]
    wchar_t* certChainData;
  GUID nonce;
  PTSG_PACKET_VERSIONCAPS versionCaps;
} TSG_PACKET_QUARENC_RESPONSE,
 *PTSG_PACKET_QUARENC_RESPONSE;
```

flags: Unused. MUST be 0.

certChainLen: An unsigned long specifying the number of characters in **certChainData**, including the terminating null character. If the **quarantineCapabilities** field of the **TSG_PACKET_VERSIONCAPS** structure is set to 1, this must be a nonzero value. This field must be ignored if **certChainData** is ignored. The value must be in the range of 0 and 24000; both inclusive.

certChainData: The certificate, along with the chain, that the TSG server used for the SCHANNEL authentication service as part of registering the RPC interfaces and initialization. It must be a string representation of the certificate chain if **certChainLen** is nonzero.<17> This field can be ignored.

nonce: A **GUID** to uniquely identify this connection to prevent replay attacks by the TSG client. This can be used for auditing purposes. A GUID is a unique ID using opaque sequence of bytes as specified in [MS-DTYP] section 2.3.2.2.

versionCaps: A **PTSG_PACKET_VERSIONCAPS** structure, as specified in section 2.2.3.2.1.2.

2.2.3.2.1.7 TSG_PACKET_CAPS_RESPONSE

The TSG_PACKET_CAPS_RESPONSE structure contains the response of the TSG server, which supports Consent Signing capability, to the TSG client for the **TsProxyCreateTunnel** method call. This structure contains **TSG_PACKET_QUARENC_RESPONSE** followed by the consent signing string. The value of the **packetId** field in **TSG_PACKET** MUST be set to **TSG_PACKET_TYPE_CAPS_RESPONSE**.

```
typedef struct TSG_PACKET_CAPS_RESPONSE {
  TSG_PACKET_QUARENC_RESPONSE pktQuarEncResponse;
  TSG_PACKET_MSG_RESPONSE pktConsentMessage;
} TSG_PACKET_CAPS_RESPONSE,
 *PTSG_PACKET_CAPS_RESPONSE;
```

pktQuarEncResponse: A **TSG_PACKET_QUARENC_RESPONSE** structure as specified in section 2.2.3.2.1.6.

pktConsentMessage: A **TSG_PACKET_MSG_RESPONSE** structure as specified in section 2.2.3.2.1.9.

2.2.3.2.1.8 TSG_PACKET_MSG_REQUEST

The **TSG_PACKET_MSG_REQUEST** structure contains the request from the client to the TSG server to send across an administrative message whenever there is any. The value of the **packetId** field in **TSG_PACKET** MUST be set to **TSG_PACKET_TYPE_MSGREQUEST_PACKET**.

```
typedef struct TSG_PACKET_MSG_REQUEST {
  unsigned long maxMessagesPerBatch;
} TSG_PACKET_MSG_REQUEST,
 *PTSG_PACKET_MSG_REQUEST;
```

maxMessagesPerBatch: An unsigned long that specifies how many messages can be sent by the server at one time.

2.2.3.2.1.9 TSG_PACKET_MSG_RESPONSE

TSG_PACKET_MSG_RESPONSE structure contains the response of the TSG server to the client when a message needs to be sent to the client. The value of the **packetId** field in **TSG_PACKET** MUST be set to **TSG_PACKET_TYPE_MESSAGE_PACKET**.

```
typedef struct _TSG_PACKET_MSG_RESPONSE {
  unsigned long msgID;
  unsigned long msgType;
  long isMsgPresent;
  [switch_is(msgType)] TSG_PACKET_TYPE_MESSAGE_UNION messagePacket;
} TSG_PACKET_MSG_RESPONSE,
 *PTSG_PACKET_MSG_RESPONSE;
```

msgID: This field is unused.<18> This field can be ignored.

msgType: An unsigned long specifying what type of message is being sent by the server. This MUST be one of the following values.

Value	Meaning
TSG_ASYNC_MESSAGE_CONSENT_MESSAGE 0x00000001	The server is sending a **Consent Signing Message**.

Value	Meaning
TSG_ASYNC_MESSAGE_SERVICE_MESSAGE 0x00000002	The server is sending an Administrative Message.
TSG_ASYNC_MESSAGE_REAUTH 0x00000003	The server expects the client to Reauthenticate.

isMsgPresent: A Boolean that indicates whether the *messagePacket* parameter is present or not. If the value is TRUE, then *messagePacket* contains valid data and may be processed. If the value is FALSE, *messagePacket* parameter must be ignored.

messagePacket: A **TSG_PACKET_TYPE_MESSAGE_UNION** union, as specified in section 2.2.3.2.1.9.1.

2.2.3.2.1.9.1 TSG_PACKET_TYPE_MESSAGE_UNION

The **TSG_PACKET_TYPE_MESSAGE_UNION** union contains the actual message that is sent by the TS Gateway server to the client. The exact type of message depends on **msgType** field as specified in section 2.2.3.2.1.9.

```
typedef
[switch_type(unsigned long)]
  union {
  [case(TSG_ASYNC_MESSAGE_CONSENT_MESSAGE)]
    PTSG_PACKET_STRING_MESSAGE consentMessage;
  [case(TSG_ASYNC_MESSAGE_SERVICE_MESSAGE)]
    PTSG_PACKET_STRING_MESSAGE serviceMessage;
  [case(TSG_ASYNC_MESSAGE_REAUTH)]
    PTSG_PACKET_REAUTH_MESSAGE reauthMessage;
} TSG_PACKET_TYPE_MESSAGE_UNION,
 *PTSG_PACKET_TYPE_MESSAGE_UNION ;
```

consentMessage: A **PTSG_PACKET_STRING_MESSAGE** structure, as defined in section 2.2.3.2.1.9.1.1. This field is used if **msgType** field specified in section 2.2.3.2.1.9 is set to **TSG_ASYNC_MESSAGE_CONSENT_MESSAGE**.

serviceMessage: A **PTSG_PACKET_STRING_MESSAGE** structure, as defined in section 2.2.3.2.1.9.1.1. This field is used if **msgType** field specified in section 2.2.3.2.1.9 is set to **TSG_ASYNC_MESSAGE_SERVICE_MESSAGE**.

reauthMessage: A **PTSG_PACKET_REAUTH_MESSAGE** structure, as defined in section 2.2.3.2.1.9.1.2. This field is used if **msgType** field specified in section 2.2.3.2.1.9 is set to **TSG_ASYNC_MESSAGE_REAUTH**.

2.2.3.2.1.9.1.1 TSG_PACKET_STRING_MESSAGE

The **TSG_PACKET_STRING_MESSAGE** structure contains either the Consent Signing Message or the Administrative Message that is being sent from the TSG server to the client.

```
typedef struct TSG_PACKET_STRING_MESSAGE {
  long isDisplayMandatory;
  long isConsentMandatory;
  [range(0,65536)] unsigned long msgBytes;
  [size_is(msgBytes)] wchar_t* msgBuffer;
} TSG_PACKET_STRING_MESSAGE,
 *PTSG_PACKET_STRING_MESSAGE;
```

isDisplayMandatory: A Boolean that specifies whether the client needs to display this message.

isConsentMandatory: A Boolean that specifies whether the user needs to give its consent before the connection can proceed.

msgBytes: An unsigned long specifying the number of characters in **msgBuffer**, including the terminating null character. The value MUST be within 0 to 65536.

msgBuffer: An array of wchar_t specifying the string. The size of the buffer is as indicated by **msgBytes**.

2.2.3.2.1.9.1.2 TSG_PACKET_REAUTH_MESSAGE

The **TSG_PACKET_REAUTH_MESSAGE** is sent by the TSG server to the client when the server requires the user credential to be reauthenticated.

```
typedef struct TSG_PACKET_REAUTH_MESSAGE {
  unsigned __int64 tunnelContext;
} TSG_PACKET_REAUTH_MESSAGE,
 *PTSG_PACKET_REAUTH_MESSAGE;
```

tunnelContext: A unsigned __int64 that is sent by the server to client. When the client initiates the reauthentication sequence, it MUST include this context. This is used by the server to validate successful reauthentication by the client.

2.2.3.2.1.10 TSG_PACKET_AUTH

The **TSG_PACKET_AUTH** structure is sent by the client to the TS Gateway server when Pluggable Authentication is used. This packet includes **TSG_PACKET_VERSIONCAPS**, which is used for capability negotiation, and cookie, which is used for user authentication. This must be the first packet from the client to the server if the server has Pluggable Authentication turned on. The value of the **packetId** field in **TSG_PACKET** MUST be set to **TSG_PACKET_TYPE_AUTH**.

```
typedef struct _TSG_PACKET_AUTH {
  TSG_PACKET_VERSIONCAPS tsgVersionCaps;
  [range(0,65536)] unsigned long cookieLen;
  [size_is(cookieLen)] byte* cookie;
} TSG_PACKET_AUTH,
 *PTSG_PACKET_AUTH;
```

tsgVersionCaps: A **TSG_PACKET_VERSIONCAPS** structure as specified in section 2.2.3.2.1.2.

cookieLen: An unsigned long that specifies the size in bytes for the field cookie.

cookie: A byte pointer that points to the cookie data. The cookie is used for authentication.

2.2.3.2.1.11 TSG_PACKET_REAUTH

The **TSG_PACKET_REAUTH** structure is sent by the client to the TS Gateway server when the client is reauthenticating the connection. The value of the **packetId** field in TSG_PACKET MUST be set to TSG_PACKET_TYPE_REAUTH.

```
typedef struct TSG_PACKET_REAUTH {
  unsigned __int64 tunnelContext;
  unsigned long packetId;
  [switch_is(packetId)] TSG_INITIAL_PACKET_TYPE_UNION tsgInitialPacket;
} TSG_PACKET_REAUTH,
 *PTSG_PACKET_REAUTH;
```

tunnelContext: An unsigned __int64 that identifies which tunnel is being reauthenticated.

packetId: An unsigned long that specifies what type of packet is present inside tsgInitialPacket.

Value	Meaning
TSG_PACKET_TYPE_VERSIONCAPS 0x00005643	This packet is sent when Pluggable Authentication is off.
TSG_PACKET_TYPE_AUTH 0x00004054	This packet is sent when Pluggable Authentication is on. This packet includes TSG_PACKET_VERSIONCAPS as well as the cookie that is required for authentication.

tsgInitialPacket: A TSG_INITIAL_PACKET_TYPE_UNION union as specified in section 2.2.3.2.1.11.1.

2.2.3.2.1.11.1 TSG_INITIAL_PACKET_TYPE_UNION

The **TSG_INITIAL_PACKET_TYPE_UNION** union is sent by the client to the TS Gateway server when the client is reauthenticating the connection. Depending on **packetId** as specified in section 2.2.3.2.1.11, either TSG_PACKET_VERSIONCAPS or TSG_PACKET_AUTH is included.

```
typedef
[switch_type(unsigned long)]
  union {
  [case(TSG_PACKET_TYPE_VERSIONCAPS)]
    PTSG_PACKET_VERSIONCAPS packetVersionCaps;
  [case(TSG_PACKET_TYPE_AUTH)]
    PTSG_PACKET_AUTH packetAuth;
} TSG_INITIAL_PACKET_TYPE_UNION,
 *PTSG_INITIAL_PACKET_TYPE_UNION;
```

packetVersionCaps: A **PTSG_PACKET_VERSIONCAPS** structure as specified in section 2.2.3.2.1.2.

packetAuth: A **PTSG_PACKET_AUTH** structure as specified in section 2.2.3.2.1.10.

2.2.3.3 Generic Send Data Message Packet

This packet contains data sent by the TSG client to the TSG server which is then sent to the target server. This is sent by the TSG client for the **TsProxySendToServer** method call.

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
PCHANNEL_CONTEXT_HANDLE_NOSERIALIZE_NR																															
...																															
...																															
...																															
...																															
totalDataBytes																															
numBuffers																															
buffer1Length																															
buffer2Length (optional)																															
buffer3Length (optional)																															
buffer1 (variable)																															
...																															
buffer2 (variable)																															
...																															
buffer3 (variable)																															
...																															

PCHANNEL_CONTEXT_HANDLE_NOSERIALIZE_NR (20 bytes): This MUST be the network representation of the **PCHANNEL_CONTEXT_HANDLE_NOSERIALIZE** data type returned by the TSG server by using the **TsProxyCreateChannel** method call. Network representation of a context handle is described in [C706] Appendix N.

totalDataBytes (4 bytes): An **unsigned long** that specifies the total number of bytes to be sent to the target server. This MUST be in network order representation. It MUST be the sum of **buffer1Length**, **buffer2Length**, and **buffer3Length** and the size of the data, in bytes, for **buffer1Length**, **buffer2Length**, and **buffer3Length**. It MUST NOT be zero.

numBuffers (4 bytes): An **unsigned long** that specifies the total number of data buffers that follow this field. This MUST be in a network order representation.

buffer1Length (4 bytes): An **unsigned long** specifying the length of the first buffer. This MUST be in a network order representation and be non-zero.

buffer2Length (4 bytes): An **unsigned long** specifying the length of the second buffer. This MUST be in a network order representation. This is optional and can be 0.

buffer3Length (4 bytes): An **unsigned long** specifying the length of the third buffer. This MUST be in a network order representation. This is optional and can be 0.

buffer1 (variable): The **buffer1** is an array of bytes. Its length is specified by **buffer1Length**. This MUST be non-NULL and contain the same number of bytes specified by **buffer1Length**. The contents of **buffer1** are opaque to the Terminal Services Gateway Server Protocol.

buffer2 (variable): The **buffer2** is an array of bytes. Its length is specified by **buffer2Length**. This MUST be non-NULL if buffer2Length is non-zero and contain the same number of bytes specified by **buffer2Length**. If buffer2Length is 0, this SHOULD be NULL. If **buffer2Length** is zero and **buffer2** is non-NULL, then **buffer2** must be ignored. The contents of **buffer2** are opaque to the Terminal Services Gateway Server Protocol.

buffer3 (variable): The **buffer3** is an array of bytes. Its length is specified by **buffer3Length**. This MUST be non-NULL if buffer3Length is nonzero and contain the same number of bytes specified by **buffer3Length**. If buffer3Length is 0, this SHOULD be NULL. If **buffer3Length** is zero and **buffer3** is non-NULL, then **buffer3** must be ignored. The contents of **buffer3** are opaque to the Terminal Services Gateway Server Protocol.

2.2.3.4 Generic Receive Pipe Message Packet

The Generic Receive Pipe Message packet has dual purposes. The packet is used by both the TSG client for setting up the receive pipe and the TSG server to send the data that is received from the target server to the TSG client.

The TSG client sends this packet in the TsProxySetupReceivePipe (section 3.1.4.2.2) method to set up the receive pipe between the TSG server and the TSG client.

The packet has three different formats in various phases as explained in the following sections.

2.2.3.4.1 TSG Client to TSG Server Packet Format

The TSG client sends the packet to the TSG server in the format below.

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
PCHANNEL_CONTEXT_HANDLE_NOSERIALIZE_NR																															
...																															
...																															
...																															

...

PCHANNEL_CONTEXT_HANDLE_NOSERIALIZE_NR (20 bytes): This MUST be the network representation of the **PCHANNEL_CONTEXT_HANDLE_NOSERIALIZE (section 2.2.1.3)** data type returned by the TSG server obtained by using the **TsProxyCreateChannel (section 3.1.4.1.4)** method call. Network representation of a context handle is described in [C706] Appendix N.

2.2.3.4.2 TSG Server to TSG Client Packet Format for Intermediate Responses

The TSG server to TSG client Packet Format for Intermediate Responses is the intermediate responses from the TSG server to the TSG client.

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
Data (variable)																															
...																															

Data (variable): This is data that the TSG server received from the target server and forwards to the TSG client. The size of this data is in the RPC headers' **alloc_hint** field specified in [C706]. Only the TSG server uses the **Data** field. This field MUST NOT be sent by the TSG client.

2.2.3.4.3 TSG Server to TSG Client Packet Format for Final Response

This is the final response from the TSG server to the TSG client. To indicate connection disconnect, TSG server MUST set the PFC_LAST_FRAG bit in pfc_flags of the header of the RPC response PDU as described in **TsProxySetupReceivePipe (section 3.1.4.2.2)**. For a description of RPC response PDU, pfc_flags, PFC_LAST_FRAG, and stub data, refer to sections 12.6.2 and 12.6.4.10 in [C706]. PDU body contains the return value as shown in the following packet diagram.

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
ReturnValue																															

ReturnValue (4 bytes): Return value of the **TsProxySetupReceivePipe** (section 3.1.4.2.2) method call.

3 Protocol Details

The following sections specify details of the Terminal Services Gateway Server Protocol, including abstract data models, interface method syntax, and message processing rules.

3.1 TsProxyRpcInterface Server Details

The following sections contain the details of the TsProxyRpcInterface on the Server.

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 that described in this document.

Target server names: An array of alias names for a target server. A target server alias name is a string of **Unicode** characters. The server name applies to the machine that the TSG server connects to.<19> This is initialized by the TSG server when the TSG client calls **TsProxyCreateChannel**. This data is passed by the TSG client in the structure **TSENDPOINTINFO**. An array of *resourceName* and *alternateResourceNames* of **TSENDPOINTINFO** structure makes target server alias names. The TSG server attempts to connect to the target server by each target server alias name until it succeeds or until the array is traversed completely.

Tunnel id: An unsigned long representing the tunnel identifier for tracking purposes on the TSG server. This is generated after a client call to **TsProxyCreateTunnel**. The Tunnel id, which is then generated on the server, is stored by the TSG server and TSG client, and can later be used for subsequent tunnel-related operations.<20> The Tunnel id is created by the **TsProxyCreateTunnel** method and points to a binary large object (BLOB) that stores the ADM elements Tunnel Context handle, Channel id, Nonce, and Number of Connections.

Channel id: An unsigned long representing the channel identifier for tracking purposes on the TSG server. This is generated after a client call to TsProxyCreateChannel. The Channel id, which is then generated on the server, is stored by the TSG server and TSG client and may later be used for subsequent channel-related calls.<21> The Channel id points to a BLOB that is created by the **TsProxyCreateChannel** method and that stores the target server name and Channel Context handle ADM element.

Tunnel Context handle: An RPC context handle for the TSG client to TSG server connection represented by an array of 20 bytes on the TSG server.

Channel Context handle: An RPC context handle for the connection from the TSG client to the target server via a TSG server represented by an array of 20 bytes on the TSG server.

Nonce: A unique GUID created by the TSG server to identify the current connection. This is used to prevent statement of health (SoH) replay attacks.

Number of Connections: An unsigned long representing the number of active connections the TSG server is processing. This is incremented on every successful call to **TsProxyCreateTunnel** and decremented on **TsProxyCloseTunnel** call.

Re-authentication Connection: A Boolean value representing whether the current connection is a normal connection or a re-authentication connection.

Re-authentication Tunnel Context: A ULONGLONG value representing a unique connection identifier. For normal connections, this value represents the unique connection identifier of the same connection. For a re-authentication connection, this value represents the unique connection identifier of a connection that has initiated the re-authentication request.

Re-authentication Status: An enumeration value representing the re-authentication status of the connection that has initiated the re-authentication.

Note Only normal connections can initiate re-authentication. Re-authentication connections cannot initiate re-authentication.

Possible values are defined in the table below.

Enumeration Value	Description
None	No progress made on the re-authentication.
AuthenticationCompleted	User authentication is done.
UserAuthorizationCompleted	User authorization is done, and if the TSG server is configured for quarantine, the TSG client is quarantine compliant.
UserAuthorizationCompletedButQurantineFailed	User authorization is done, and the TSG server is configured for quarantine but the TSG client is not quarantine compliant.
ResourceAuthorizationCompleted	Resource authorization is done. If **Re-authentication Status** reaches this state, it means that re-authentication is completed.

This ADM element is valid only for normal connection, that is, when **Re-authentication Connection** is FALSE.

Negotiated Capabilities: A ULONG bitmask value representing the negotiated capabilities between the TSG client and the TSG server. It contains zero or more of the following NAP Capability values:

NAP Capability Value
TSG_NAP_CAPABILITY_QUAR_SOH
TSG_NAP_CAPABILITY_IDLE_TIMEOUT
TSG_MESSAGING_CAP_CONSENT_SIGN
TSG_MESSAGING_CAP_SERVICE_MSG
TSG_MESSAGING_CAP_REAUTH

3.1.1.1 TSG Server States

Connection State: An enum of different connection states. This is updated as per the state transition rules mentioned in section 3.1.4. The following diagram represents the connection state transition.

The TSG server MUST use this ADM element to verify that the call sequence is not violated. In each state the allowed calls and the state transitions therefore are described in this section. Section 3.1.4 describes the returns values and errors for each method call.

Each connection goes through a set of states as described in this section.

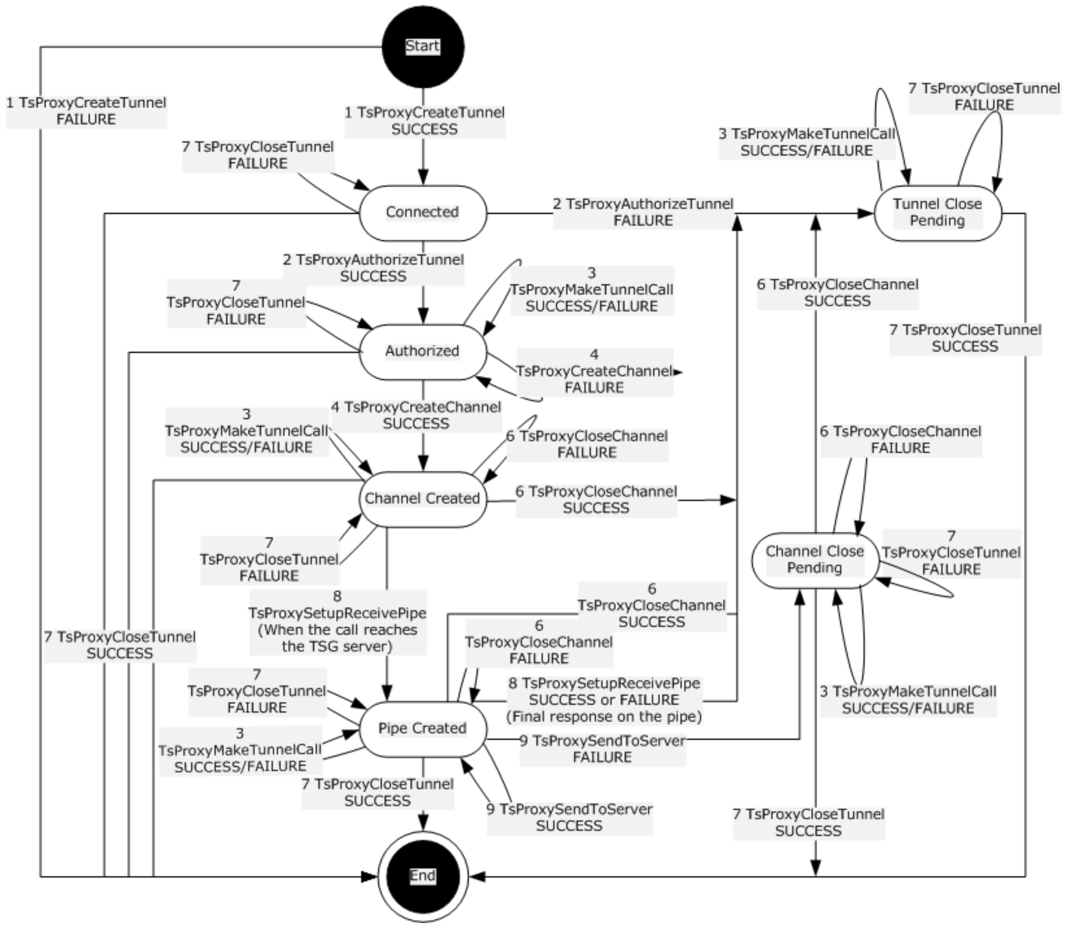
Start
1 TsProxyCreateTunnel FAILURE
1 TsProxyCreateTunnel SUCCESS
7 TsProxyCloseTunnel FAILURE
Connected
2 TsProxyAuthorizeTunnel FAILURE
3 TsProxyMakeTunnelCall SUCCESS/FAILURE
7 TsProxyCloseTunnel FAILURE
Tunnel Close Pending
2 TsProxyAuthorizeTunnel SUCCESS
7 TsProxyCloseTunnel FAILURE
3 TsProxyMakeTunnelCall SUCCESS/FAILURE
6 TsProxyCloseChannel SUCCESS
7 TsProxyCloseTunnel SUCCESS
Authorized
4 TsProxyCreateChannel FAILURE
3 TsProxyMakeTunnelCall SUCCESS/FAILURE
4 TsProxyCreateChannel SUCCESS
6 TsProxyCloseChannel FAILURE
6 TsProxyCloseChannel FAILURE
Channel Created
6 TsProxyCloseChannel SUCCESS
7 TsProxyCloseTunnel FAILURE
Channel Close Pending
7 TsProxyCloseTunnel FAILURE
8 TsProxySetupReceivePipe (When the call reaches the TSG server)
6 TsProxyCloseChannel SUCCESS
7 TsProxyCloseTunnel SUCCESS
6 TsProxyCloseChannel FAILURE
7 TsProxyCloseTunnel FAILURE
8 TsProxySetupReceivePipe SUCCESS or FAILURE (Final response on the pipe)
3 TsProxyMakeTunnelCall SUCCESS/FAILURE
Pipe Created
9 TsProxySendToServer FAILURE
3 TsProxyMakeTunnelCall SUCCESS/FAILURE
7 TsProxyCloseTunnel SUCCESS
9 TsProxySendToServer SUCCESS
7 TsProxyCloseTunnel SUCCESS
End

Figure 5: State transition diagram

Start: By default, the protocol starts in a disconnected state.

Connected: A successful **TsProxyCreateTunnel** call brings the connection to the Connected state. Once a connection is in a Connected state, a **TsProxyCloseTunnel** call can be made to bring the connection to the Disconnected state.

Authorized: A successful **TsProxyAuthorizeTunnel** call brings the connection to the Authorized state. A **TsProxyAuthorizeTunnel** call can only be made when the connection is in a Connected state. If a **TsProxyAuthorizeTunnel** call is made in any other state, then the result is undefined. The **TsProxyMakeTunnelCall** call is allowed in this state. This call does not change the state. **TsProxyCloseTunnel** can also be made in this state, which moves the connection to the Disconnected state.

Channel Created: A successful **TsProxyCreateChannel** call brings the connection to the Channel Created state. A **TsProxyCreateChannel** call is valid only when the tunnel is authorized. If a

TsProxyCreateChannel call is made before the tunnel is authorized, ERROR_ACCESS_DENIED will be returned. **TsProxyCloseChannel** can also be made in this state, which moves the connection to the Tunnel Close Pending state. The **TsProxySetupReceivePipe** call is valid only in this state. If this call is made before the TSG client calls **TsProxyCreateChannel**, ERROR_ACCESS_DENIED will be returned. If it is made after the call to **TsProxyCloseChannel**, E_PROXY_ALREADYDISCONNECTED will be returned.

The TsProxyMakeTunnelCall call is allowed in this state. This call does not change the state.

When **TsProxyCloseTunnel** is called in this state before a call to **TsProxyCloseChannel**, the TSG server closes the channel and completes the **TsProxyCloseTunnel** call. After completing this call, the TSG server moves to the End state.

Pipe Created: When a call to **TsProxySetupReceivePipe** reaches the TSG server, the connection goes to the Pipe Created state. The **TsProxySendToServer** call is valid only in this state. If this call is made before the TSG client calls **TsProxySetupReceivePipe**, ERROR_ACCESS_DENIED will be returned. If it is made after the call to **TsProxyCloseTunnel**, E_PROXY_ALREADYDISCONNECTED will be returned.

The TsProxyMakeTunnelCall call is allowed in this state. This call does not change the state.

When **TsProxyCloseTunnel** is called in this state before a call to **TsProxyCloseChannel**, the TSG server closes the channel and completes the **TsProxyCloseTunnel** call. After completing this call, the TSG server moves to the End state.

Channel Close Pending: From Pipe Created state, either a final response to **TsProxySetupReceivePipe** call or a failure in **TsProxySendToServer** call brings the connection to the Channel Close Pending state. **TsProxyCloseChannel**, TsProxyMakeTunnelCall, and **TsProxyCloseTunnel** calls are the only valid calls in this state.

When **TsProxyCloseTunnel** is called in this state before a call to **TsProxyCloseChannel**, the TSG server closes the channel and completes the **TsProxyCloseTunnel** call. After completing this call, the TSG server moves to the End state.

Tunnel Close Pending: Either a failure **TsProxyAuthorizeTunnel** call from Connected state or a successful **TsProxyCloseChannel** call from Channel Close Pending state brings the connection to Tunnel Close Pending state. If a previous **TsProxyMakeTunnelCall** has not completed, then another call to **TsProxyMakeTunnelCall** MUST be made as specified in section 3.1.4.3.2. The **TsProxyCloseTunnel** call SHOULD be made by the TSG client to end the protocol.

End: The TSG server MUST transition to this state when the **TsProxyCloseTunnel** method is called. At this stage, the connection between the TSG client and the TSG server is disconnected.

3.1.2 Timers

3.1.2.1 Session Timeout Timer

After a successful call to the **TsProxyCreateChannel** method, if session timeout is configured at the TSG server,<22> the TSG server MUST start this timer with the configured session-timeout value. Default value of the timer is zero, which means no session timeout. The timeout value MUST be between 0 and 4294967295 in units of minutes.

3.1.2.2 Re-authentication Timer

Default value of the timer is 1 minute.<23> The time value MUST be between 1 and 3, both inclusive, in units of minutes. The TSG server MUST start this timer after it sends the re-authentication message to the TSG client.

3.1.2.3 Connection Timer

The TSG server MAY use this timer to recover early instead of waiting for long periods for a successful connection to the target server.<24>

Default value of the timer is 30 seconds.<25> The timer value MUST be between 30 seconds and 3 minutes, both inclusive, in units of minutes. This timer MUST be started after the call to **TsProxyCreateChannel** is received by the TSG server.

If a call to **TsProxySetupReceivePipe** is received by the TSG server before the timer expires, the timer MUST be stopped.

If the call to the **TsProxySetupReceivePipe** is received by the TSG server after the timer has expired, the server MUST disconnect with the ERROR_OPERATION_ABORTED return value, as specified in section 2.2.2.24.

3.1.3 Initialization

The protocol uses the transport and endpoints as described in section 2.1.

The initialization steps for the TSG server are as follows: The TSG server MUST register for ipv4 and ipv6 local host addresses 127.0.0.1 and ::1 as the network address when operating in a non-load balanced environment. The TSG server MUST register for RPC_C_AUTHN_GSS_NEGOTIATE and SHOULD register for RPC_C_AUTHN_GSS_SCHANNEL as authentication services, as specified in [MS-RPCE] section 2.2.1.1.7. The TSG client MUST use a minimum authentication level of RPC_C_AUTHN_LEVEL_PKT_INTEGRITY (see [MS-RPCE] section 2.2.1.1.8) and MUST use one of the following authentication services: RPC_C_AUTHN_GSS_NEGOTIATE or RPC_C_AUTHN_GSS_SCHANNEL, or RPC_C_AUTHN_WINNT.<26>

All timers are connection-specific timers, and MUST not be started on initialization.

3.1.4 Message Processing Events and Sequencing Rules

This protocol asks the RPC runtime to perform a strict NDR data consistency check at target level 7.0 for all methods unless otherwise specified, as specified in [MS-RPCE] section 3.

The TSG server SHOULD<27> enforce appropriate security measures to be sure that the caller has the required permissions to execute the following routines.

The methods MAY throw an exception, and the client MUST handle these exceptions gracefully. The methods implemented by the TSG server MUST be sequential in order as specified in section 1.3.1. The method details are specified as follows.

Methods in RPC Opnum Order

Method	Description
Opnum0NotUsedOnWire	Reserved for local use. Opnum: 0

Method	Description
TsProxyCreateTunnel	Sets up the context in which all further communication between the TSG client and the TSG server occurs. Opnum: 1
TsProxyAuthorizeTunnel	Authorizes the tunnel based on rules defined by the TSG server. Opnum: 2
TsProxyMakeTunnelCall	Used to request for administrative messages from the TSG server when the same are available. This method is only called when both the client and the TSG server are capable of handling administrative messages. The request is queued up on the TSG server. The same method is also called during shutdown sequence to cancel any pending administrative message request.<28> Opnum: 3
TsProxyCreateChannel	Creates a channel between the TSG client and the target server via the TSG server that the TSG client desires to connect. Opnum: 4
Opnum5NotUsedOnWire	Reserved for local use. Opnum: 5
TsProxyCloseChannel	Closes the channel between the TSG client and the target server. Opnum: 6
TsProxyCloseTunnel	Closes the tunnel between the TSG client and the TSG server. Opnum: 7
TsProxySetupReceivePipe	Used for data transfer from the TSG server to the TSG client. Opnum: 8
TsProxySendToServer	Used for data transfer from the TSG client to the TSG server. Opnum: 9

Note In the preceding table, the term "Reserved for local use" means that the client MUST NOT send the opnum, and the TSG server behavior is undefined<29> because it does not affect interoperability.

3.1.4.1 Connection Setup Phase

3.1.4.1.1 TsProxyCreateTunnel (Opnum 1)

The **TsProxyCreateTunnel** method sets up the tunnel in which all further communication between the TSG client and the TSG server occurs. This is also used to exchange versioning and capability information between the TSG client and TSG server. It is used to exchange the TSG server certificate which has already been used to register for an authentication service. After this method call has successfully been completed, a tunnel shutdown can be performed. This is accomplished by using the **TsProxyCloseTunnel** method call.

Prerequisites: The connection state MUST be in Start state.

Sequential Processing Rules:

1. If any unexpected error occurs in the below process, the TSG server MUST return E_PROXY_INTERNALERROR.
2. The TSG server MUST verify that a server authentication certificate is registered with SCHANNEL authentication service. Otherwise it MUST return E_PROXY_NOCERTAVAILABLE.
3. If the TSG server is configured for pluggable authentication:
    1. The TSG server MUST verify that the **packetId** member of the *tsgPacket* parameter is either **TSG_PACKET_TYPE_AUTH** or **TSG_PACKET_TYPE_REAUTH**. Otherwise, it MUST return the E_PROXY_UNSUPPORTED_AUTHENTICATION_METHOD error code.
    2. If the **packetId** member of *tsgPacket* parameter is **TSG_PACKET_TYPE_AUTH**, then the TSG server MUST verify that tsgPacket->tsgPacket.packetAuth is not NULL and tsgPacket->tsgPacket.packetAuth->cookie is not NULL and tsgPacket->tsgPacket.packetAuth->cookieLen is not zero. Otherwise, it MUST return E_PROXY_COOKIE_BADPACKET. If the **packetId** member of the *tsgPacket* parameter is **TSG_PACKET_TYPE_REAUTH**, then the TSG server MUST verify that tsgPacket->tsgPacket.packetReauth->tsgInitialPacket.packetAuth is not NULL and tsgPacket->tsgPacket.packetReauth->tsgInitialPacket.packetAuth->cookie is not NULL and tsgPacket->tsgPacket.packetReauth->tsgInitialPacket.packetAuth->cookieLen is not zero. Otherwise, it MUST return E_PROXY_COOKIE_BADPACKET.
    3. The TSG server MUST authenticate the user using the cookie. If authentication fails, it MUST return E_PROXY_COOKIE_AUTHENTICATION_ACCESS_DENIED error code.
4. If the TSG server is configured for RPC authentication:
    1. The TSG server MUST verify that the **packetId** member of the *tsgPacket* parameter type is either **TSG_PACKET_TYPE_VERSIONCAPS** or **TSG_PACKET_TYPE_REAUTH**. Otherwise, it MUST return the E_PROXY_INTERNALERROR error code.
5. The TSG server MUST create a GUID and initialize the ADM element **Nonce** with it.
6. The TSG server MUST create a unique identifier and initialize the ADM element **Tunnel Id** with it.
7. If the **packetId** member of the *tsgPacket* parameter type is not **TSG_PACKET_TYPE_REAUTH**:
    1. The TSG server MUST initialize the ADM element **Re-authentication Connection** to FALSE.
    2. The TSG server MUST initialize the ADM element **Re-authentication Status** to NONE.
    3. The TSG server MUST initialize the ADM element **Re-authentication Tunnel Context** with a unique ULONGLONG identifier. This identifier MUST be used by the re-authentication connection to find this connection and set its **Re-authentication Status** ADM element.
8. If the **packetId** member of the *tsgPacket* parameter is **TSG_PACKET_TYPE_REAUTH**:
    1. The TSG server MUST initialize the ADM element **Re-authentication Connection** to TRUE.
    2. The TSG server MUST not use the ADM element **Re-authentication Status** for this connection.
    3. The TSG server MUST initialize the ADM element **Re-authentication Tunnel Context** with tsgPacket->tsgPacket.packetReauth->tunnelContext.

4. The TSG server MUST find the original connection that has initiated the re-authentication using **Re-authentication Tunnel Context**, and its ADM element **Re-authentication Status** MUST be set to AuthenticationCompleted.

9. The TSG server MUST create a tunnel context handle and MUST initialize the ADM element **Tunnel Context Handle** with it.

10. The TSG server MUST initialize the ADM element **Negotiated Capabilities** with the common capabilities between the TSG client and the TSG server.

11. If the TSG server supports the TSG_MESSAGING_CAP_CONSENT_SIGN capability and is configured to allow only a TSG client that supports the TSG_MESSAGING_CAP_CONSENT_SIGN capability, but the TSG client doesn't support the capability, then the TSG server MUST return the E_PROXY_CAPABILITYMISMATCH error.

12. If the ADM element **Negotiated Capabilities** contains the **TSG_MESSAGING_CAP_CONSENT_SIGN** value, the **packetId** member of the *tsgPacketResponse* out parameter MUST be TSG_PACKET_TYPE_CAPS_RESPONSE. Otherwise, the **packetId** member of *tsgPacketResponse* MUST be TSG_PACKET_TYPE_QUARENC_RESPONSE.

13. The TSG server MUST set the **certChainData** field of TSG_PACKET_QUARENC_RESPONSE structure in *tsgPacketResponse* only when quarantine is configured at the TSG server and the ADM element **Negotiated Capabilities** contains TSG_NAP_CAPABILITY_QUAR_SOH.

14. The TSG server MUST return ERROR_SUCCESS.

```
HRESULT TsProxyCreateTunnel(
  [in, ref] PTSG_PACKET tsgPacket,
  [out, ref] PTSG_PACKET* tsgPacketResponse,
  [out] PTUNNEL_CONTEXT_HANDLE_SERIALIZE* tunnelContext,
  [out] unsigned long* tunnelId
);
```

tsgPacket: Pointer to the TSG_PACKET structure. If this call is made for a re-authentication, then the **packetId** field MUST be set to TSG_PACKET_TYPE_REAUTH and the **packetReauth** field of the *tsgPacket* union field MUST be a pointer to the TSG_PACKET_REAUTH structure. Otherwise, if this call is made for a new connection and the TSG server is configured for RPC authentication, then the value of the **packetId** field MUST be set to **TSG_PACKET_TYPE_VERSIONCAPS** and the **packetVersionCaps** field of the *tsgPacket* union field MUST be a pointer to the TSG_PACKET_VERSIONCAPS structure. Otherwise, if this call is made for a new connection and the TSG server is configured for pluggable authentication<30>, then the value of the **packetId** field MUST be set to **TSG_PACKET_TYPE_AUTH** and the **packetAuth** field of the *tsgPacket* union field MUST be a pointer to the TSG_PACKET_AUTH structure. If **TSG_PACKET_AUTH** is not populated correctly, the error E_PROXY_COOKIE_BADPACKET is returned.<31>

tsgPacketResponse: Pointer to the **TSG_PACKET** structure. If TSG_MESSAGING_CAP_CONSENT_SIGN capability is negotiated, the **packetId** member of the *tsgPacketResponse* out parameter MUST be **TSG_PACKET_TYPE_CAPS_RESPONSE** and the **packetCapsResponse** field of the **tsgPacket** union field MUST be a pointer to the TSG_PACKET_CAPS_RESPONSE (section 2.2.3.2.1.7). Otherwise, the **packetId** member of *tsgPacketResponse* MUST be **TSG_PACKET_TYPE_QUARENC_RESPONSE**, and the **packetQuarEncResponse** field of the *tsgPacket* union field MUST be a pointer to the **TSG_PACKET_QUARENC_RESPONSE** structure. The ADM element Nonce MUST be initialized to a unique GUID and assigned to the **nonce** field of the **TSG_PACKET_QUARENC_RESPONSE** structure either in tsgPacketResponse->tsgPacket.packetQuarEncResponse or tsgPacketResponse->tsgPacket.packetCapsResponse->pktQuarEncResponse.

tunnelContext: An RPC context handle that represents context-specific information for the tunnel. The TSG server MUST provide a non-NULL value. The TSG client MUST save and use this context handle on all subsequent methods calls on the tunnel. The methods are **TsProxyAuthorizeTunnel**, **TsProxyCreateChannel**, and **TsProxyCloseTunnel**.

tunnelId: An **unsigned long** identifier representing the tunnel. The TSG server MUST save this value in the ADM element **Tunnel id** and SHOULD provide this value to the TSG client. The TSG client SHOULD save the *tunnel id* for future use on the TSG client itself. This *tunnel id* is not required on any future method calls to the TSG server; the *tunnelContext* is used instead.

Return Values: The method MUST return ERROR_SUCCESS on success. Other failures MUST be one of the codes listed in the rest of this table. The client MAY interpret failures in any way it deems appropriate. See 2.2.2.24 for details on these errors.

Return value	State transition	Description
ERROR_SUCCESS (0x00000000)	The connection MUST transition to the connected state.	Returned when a call to the **TsProxyCreateTunnel** method succeeds.
E_PROXY_INTERNALERROR (0x800759D8)	The connection MUST transition to end state.	Returned when the server encounters an unexpected error. The TSG client MUST end the protocol when this error is received.
E_PROXY_COOKIE_BADPACKET (0x800759F7)	The connection MUST transition to end state.	Returned if the **packetAuth** field of the *tsgPacket* parameter is NULL.
E_PROXY_NOCERTAVAILABLE (0x800759EE)	The connection MUST transition to end state.	Returned when the TSG server cannot find a certificate to register for SCHANNEL Authentication Service (AS). The TSG client MUST end the protocol when this error is received.
E_PROXY_UNSUPPORTED_AUTHENTICATION_METHO	The	Returned to the TSG client when

Return value	State transition	Description
D(0x800759F9)	connection MUST transition to end state.	the TSG server is configured for pluggable authentication and the value of the **packetId** member of the *tsgPacket* parameter is not equal to TSG_PACKET_TYPE_AUTH or TSG_PACKET_TYPE_REAUTH. The TSG server MUST disconnect the connection.
E_PROXY_COOKIE_AUTHENTICATION_ACCESS_DENIED (0x800759F8)	The connection MUST transition to end state.	Returned when the given user does not have access to connect via TSG server. The TSG server MUST be in pluggable authentication mode for this error to be returned.
E_PROXY_CAPABILITYMISMATCH (0x800759E9)	The connection MUST transition to end state.	Returned when the TSG server supports the **TSG_MESSAGING_CAP_CONSENT_SIGN** capability and is configured to allow only a TSG client that supports the **TSG_MESSAGING_CAP_CONSENT_SIGN** capability, but the TSG client doesn't support the capability.

3.1.4.1.2 TsProxyAuthorizeTunnel (Opnum 2)

The **TsProxyAuthorizeTunnel** method is used to authorize the tunnel based on rules defined by the TSG server. The TSG server SHOULD perform security authorization for the TSG client. The TSG server MAY also use this method to require health checks from the TSG client which MAY require the TSG client to perform health remediation.<32> After this method call has successfully been completed, a tunnel shutdown can be performed. If there are existing channels within the tunnel, the TSG server MUST close all the channels before the tunnel shutdown. The tunnel shutdown is accomplished by using the **TsProxyCloseTunnel** method call.

If this method call completes successfully, the ADM element **Number of Connections** MUST be incremented by 1.

Prerequisites: The connection MUST be in Connected state. If this call is made in any other state, the result is undefined.

Sequential Processing Rules:

1. The TSG server MUST verify that the **packetId** field of the *tsgPacket* parameter is **TSG_PACKET_TYPE_QUARREQUEST**. Otherwise, it MUST return HRESULT_CODE(E_PROXY_NOTSUPPORTED).
2. If **Negotiated Capabilities** contains **TSG_NAP_CAPABILITY_QUAR_SOH** and tsgPacket->tsgPacket.packetQuarRequest->dataLen is not zero and tsgPacket->tsgPacket.packetQuarRequest->data is not NULL, then the following.

- The TSG server MUST verify the signature of the SoHR specified in tsgPacket->tsgPacket.packetQuarRequest->data with the TSG server certificate and decode it.
- The TSG server MUST also verify that the decoded message is prefixed with the **Nonce**. Otherwise, it MUST return ERROR_INVALID_PARAMETER.
- The remaining bytes in the decoded message are the TSG client computer's statement of health response (SoHR).

3. If **Negotiated Capabilities** doesn't contain **TSG_NAP_CAPABILITY_QUAR_SOH**, then the TSG server MUST ignore tsgPacket->tsgPacket.packetQuarRequest->dataLen and tsgPacket->tsgPacket.packetQuarRequest->data.
4. The TSG server MUST verify that the ADM element **Number of Connections** has not already reached the maximum number of connections configured by the TSG service. Otherwise, it MUST return the E_PROXY_MAXCONNECTIONSREACHED error code.
5. The TSG server MUST do the user authorization as per policies configured at the TSG server. If the user is not authorized, it MUST return E_PROXY_NAP_ACCESS_DENIED.
6. If quarantine is configured at the TSG server:
    1. The TSG client computer's statement of health (SoH) SHOULD be passed to NAPSO by calling Proxy SoH Task (section 7), as specified in [MS-NAPSO] section 7, with the correct parameters.
    2. As specified in [MS-NAPSO] section 7, control is subsequently returned to the protocol after SoH processing by calling the TSGU server abstract interface and passing the result of the SoH processing, which is the statement of health response (SoHR), to the TSG server abstract interface.
    3. The TSG server MUST sign the SoHR using **SHA-1 hash** and encode it with the TSG server certificate and append the signed and encoded SoHR to tsgPacketResponse->tsgPacket.packetResponse->responseData, where *tsgPacketResponse* is an output parameter to **TsProxyAuthorizeTunnel**.
    4. If the TSG client computer's health is not compliant to quarantine settings:
        - If the ADM element **Re-authentication Connection** is TRUE:
            1. The TSG server MUST find the original connection that has initiated the re-authentication using **Re-authentication Tunnel Context** and MUST set its ADM element **Re-authentication Status** to UserAuthorizationCompletedButQuarantineFailed.
            2. The TSG server MUST return the E_PROXY_QUARANTINE_ACCESSDENIED error code.
7. If the ADM element **Re-authentication Connection** is TRUE:
    - The TSG server MUST find the original connection which has initiated the re-authentication using **Re-authentication Tunnel Context** and MUST set its ADM element **Re-authentication Status** to *UserAuthorizationCompleted*.
8. The TSG server MUST set the **packetId** member of the *tsgPacketResponse* out parameter to **TSG_PACKET_TYPE_RESPONSE**.
9. The TSG server MUST increment the ADM element **Number of Connections** by 1.
10. The TSG server MUST return ERROR_SUCCESS.

```
HRESULT TsProxyAuthorizeTunnel(
  [in] PTUNNEL_CONTEXT_HANDLE_NOSERIALIZE tunnelContext,
  [in, ref] PTSG_PACKET tsgPacket,
  [out, ref] PTSG_PACKET* tsgPacketResponse
);
```

tunnelContext: The TSG client MUST provide the TSG server with the same context handle it received from the **TsProxyCreateTunnel** method call. The TSG server SHOULD throw an exception if the RPC validation and verification fails.

tsgPacket: Pointer to the **TSG_PACKET** structure. The value of the **packetId** field MUST be set to **TSG_PACKET_TYPE_QUARREQUEST**. If this is set to any other value, the error E_PROXY_NOT_SUPPORTED is returned. The **packetQuarRequest** field of the *tsgPacket* union field MUST be a pointer to the **TSG_PACKET_QUARREQUEST** structure.

tsgPacketResponse: Pointer to the **TSG_PACKET** structure. The value of the **packetId** field MUST be **TSG_PACKET_TYPE_RESPONSE**. The **packetResponse** field of the *tsgPacket* union field MUST be a pointer to the **TSG_PACKET_RESPONSE** structure.

Return Values: The method MUST return ERROR_SUCCESS on success. Other failures MUST be one of the codes listed. The client MAY interpret failures in any way it deems appropriate. See 2.2.2.24 for details on these errors.

Return value	State transition	Description
ERROR_SUCCESS (0x00000000)	The connection MUST transition to the authorized state.	Returned when a call to the **TsProxyAuthorizeTunnel** method succeeds.
E_PROXY_NAP_ACCESSDENIED (0x800759DB)	The connection MUST transition to Tunnel Close Pending state.	Returned when the TSG server denies the TSG client access due to policy. The TSG client MUST end the protocol when this error is received.
HRESULT_CODE(E_PROXY_NOTSUPPORTED) (0x000059E8)	The connection MUST transition to Tunnel Close Pending state.	Returned if the **packetId** field of the *tsgPacket* parameter is not TSG_PACKET_TYPE_QUARREQUEST. The TSG client MUST end the protocol when this error is received.
E_PROXY_QUARANTINE_ACCESSDENIED (0x800759ED)	The connection MUST transition	Returned when the TSG server rejects the connection due to quarantine policy. The TSG client MUST end the protocol when this

Return value	State transition	Description
	to Tunnel Close Pending state.	error is received.
ERROR_ACCESS_DENIED (0x00000005)	The connection MUST transition to Tunnel Close Pending state.	Returned when this call is made either in a state other than the Connected state or the *tunnelContext* parameter is NULL. The TSG client MUST end the protocol when this error is received.
HRESULT_CODE(E_PROXY_MAXCONNECTIONSREACHED) (0x59E6)	The connection MUST transition to end state.	Returned when the ADM element **Number of Connections** is equal to the maximum number of connections when the call is made.<33> The TSG client MUST end the protocol when this error is received.
ERROR_INVALID_PARAMETER (0x00000057)	The connection MUST not transition its state.	Returned when the **Negotiated Capabilities** ADM element contains TSG_NAP_CAPABILITY_QUAR_SOH and tsgPacket->tsgPacket.packetQuarRequest->dataLen is not zero and tsgPacket->tsgPacket.packetQuarRequest->data is not NULL and tsgPacket->tsgPacket.packetQuarRequest->data is not prefixed with Nonce.

3.1.4.1.3 TsProxyMakeTunnelCall (Opnum 3)

The **TsProxyMakeTunnelCall** method is designed to be used as a general purpose API. If both the client and the server support the administrative message, the client MAY request the same from the TSG server. If the TSG server has any administrative messages, it SHOULD complete the pending call at this point in time. After a call to **TsProxyMakeTunnelCall** returns, the TSG client SHOULD queue up another request at this point in time. During the shutdown sequence, the client MUST make this call, if a request is pending on the TSG server, to cancel the administrative message request.

Prerequisites: The connection MUST be in Authorized state or Channel Created state or Pipe Created state or Channel Close Pending state or Tunnel Close Pending state. If this call is made in any other state, the error ERROR_ACCESS_DENIED is returned.

Sequential Processing Rules:

1. The TSG server MUST verify that the *procId* parameter is either **TSG_TUNNEL_CALL_ASYNC_MSG_REQUEST** or

   TSG_TUNNEL_CANCEL_ASYNC_MSG_REQUEST. Otherwise, it MUST return ERROR_ACCESS_DENIED.

2. The TSG server MUST verify that the tunnel has been authorized. Otherwise, it MUST return ERROR_ACCESS_DENIED.

3. The TSG server MUST verify that the ADM element **Re-authentication Connection** is FALSE. Otherwise, it MUST return ERROR_ACCESS_DENIED. **TsProxyMakeTunnelCall** is not valid on re-authentication tunnels.

4. If *procId* is **TSG_TUNNEL_CALL_ASYNC_MSG_REQUEST**:

   1. If a **TsProxyMakeTunnelCall** has already been made and not yet returned, the TSG server MUST return ERROR_ACCESS_DENIED.

   2. If there is already a pending **administrative message** or re-authentication message to the TSG client, the TSG server MUST fill *tsgPacketResponse* and return ERROR_SUCCESS.

   3. If there is no pending administrative message or a re-authentication message, the TSG server MUST wait until one of the following events occurs:

      - Re-authentication starts because the session timeout timer expires.

      - The TS Gateway server administrator sets the administrative message.

      - The TSG client cancels the call.

      - The connection shutdown sequence is initiated.

      If any of the preceding events occurs, then the following steps MUST be performed:

      1. If re-authentication is started because of session timeout timer expiration, then the TSG server MUST return the **TsProxyMakeTunnelCall** as explained in section 3.1.5.1.

      2. Or else, if the TS Gateway administrator has set the administrative message, then the TSG server MUST do the following:

         1. The TSG server MUST set the **packetId** member of the *tsgPacketResponse* out parameter of **TsProxyMakeTunnelCall** to **TSG_PACKET_TYPE_MESSAGE_PACKET**.

         2. The TSG server MUST set tsgPacketResponse->packetMsgResponse->msgType to **TSG_ASYNC_MESSAGE_SERVICE_MESSAGE**.

         3. The TSG server MUST initialize tsgPacketResponse->packetMsgResponse->messagePacket.serviceMessage->isDisplayMandatory to TRUE.

         4. The TSG server MUST initialize tsgPacketResponse->packetMsgResponse->messagePacket.serviceMessage->isConsentMandatory to FALSE.

         5. The TSG server MUST initialize tsgPacketResponse->packetMsgResponse->messagePacket.serviceMessage->msgBuffer with the administrative message.

         6. The TSG server MUST initialize tsgPacketResponse->packetMsgResponse->messagePacket.serviceMessage->msgBytes with the number of characters in tsgPacketResponse->packetMsgResponse->messagePacket.serviceMessage->msgBuffer.

        7. The TSG server MUST complete the **TsProxyMakeTunnelCall** with error code ERROR_SUCCESS.

    3. Or else, if the TSG client cancels the call by calling another **TsProxyMakeTunnelCall** with *procId* **TSG_TUNNEL_CANCEL_ASYNC_MSG_REQUEST**, then the TSG server MUST return HRESULT_FROM_WIN32(RPC_S_CALL_CANCELLED).

    4. Or else, if the connection shutdown sequence is initiated, then the TSG server MUST return HRESULT_FROM_WIN32(RPC_S_CALL_CANCELLED).

5. If *procId* is **TSG_TUNNEL_CANCEL_ASYNC_MSG_REQUEST**:

    1. If there is no unreturned **TsProxyMakeTunnelCall** call which is called with the *procId* value **TSG_TUNNEL_CALL_ASYNC_MSG_REQUEST**, the TSG server MUST return ERROR_ACCESS_DENIED.

    2. Otherwise, the TSG server MUST notify the waiting **TsProxyMakeTunnelCall** call, which is called with the *procId* value **TSG_TUNNEL_CALL_ASYNC_MSG_REQUEST**, that the TSG client is canceling the call.

    3. TSG server MUST return ERROR_SUCCESS.

```
HRESULT TsProxyMakeTunnelCall(
  [in] PTUNNEL_CONTEXT_HANDLE_NOSERIALIZE tunnelContext,
  [in] unsigned long procId,
  [in, ref] PTSG_PACKET tsgPacket,
  [out, ref] PTSG_PACKET* tsgPacketResponse
);
```

tunnelContext: The TSG client MUST provide the TSG server with the same context handle it received from the TsProxyCreateTunnel method call. The TSG server SHOULD throw an exception if the RPC validation and verification fail.

procId: This field identifies the work that is performed by the API. This field can have the following values.

Value	Meaning
TSG_TUNNEL_CALL_ASYNC_MSG_REQUEST 0x00000001	Used to request an administrative message when the same is available on the server.
TSG_TUNNEL_CANCEL_ASYNC_MSG_REQUEST 0x00000002	Used to cancel a pending administrative message request.

tsgPacket: Pointer to the TSG_PACKET structure. The value of the **packetId** field MUST be set to TSG_PACKET_TYPE_MSGREQUEST_PACKET. The **packetMsgRequest** field of the **tsgPacket** union field MUST be a pointer to the TSG_PACKET_MSG_REQUEST structure.

tsgPacketResponse: Pointer to the **TSG_PACKET** structure. If *procId* is **TSG_TUNNEL_CANCEL_ASYNC_MSG_REQUEST** or if the return value is HRESULT_FROM_WIN32(RPC_S_CALL_CANCELLED), **tsgPacketResponse* MUST be set to NULL. Otherwise, the value of the **packetId** field MUST be **TSG_PACKET_TYPE_MESSAGE_PACKET**. The **packetMsgResponse** field of the **tsgPacket** union field MUST be a pointer to the TSG_PACKET_MSG_RESPONSE structure.

Return Values: The method MUST return ERROR_SUCCESS on success. Other failures MUST be one of the codes listed. The client MAY interpret failures in any way it deems appropriate. See 2.2.2.24 for details on these errors. The connection MUST NOT transition its state after completing the **TsProxyMakeTunnelCall**.

Return value	State transition	Description
ERROR_SUCCESS (0x00000000)	The connection MUST NOT transition its state.	Returned when a call to the **TsProxyMakeTunnelCall** method succeeds.
ERROR_ACCESS_DENIED (0x00000005)	The connection MUST NOT transition its state.	Returned in the following cases. ▪ When the call is made in any state other than Authorized, Channel Created, Pipe Created, Channel Close Pending, or Tunnel Close Pending. ▪ If **procId** is neither TSG_TUNNEL_CALL_ASYNC_MSG_REQUEST nor TSG_TUNNEL_CANCEL_ASYNC_MSG_REQUEST. ▪ If **procId** is TSG_TUNNEL_CALL_ASYNC_MSG_REQUEST and there is already a call to **TsProxyMakeTunnelCall** made earlier with **procId** TSG_TUNNEL_CALL_ASYNC_MSG_REQUEST and it is not yet returned. ▪ If **procId** is TSG_TUNNEL_CANCEL_ASYNC_MSG_REQUEST and there is no call to **TsProxyMakeTunnelCall** made earlier with **procId** TSG_TUNNEL_CALL_ASYNC_MSG_REQUEST that is not yet returned. ▪ If the *tunnelContext* parameter is NULL. ▪ If the tunnel is not authorized. ▪ If the **Re-authentication Connection** ADM element is TRUE. The TSG client MUST end the

Return value	State transition	Description
		protocol when this error is received.
HRESULT_FROM_WIN32(RPC_S_CALL_CANCELLED)(0x8007071A)	The connection MUST not transition its state.	Returned when the call is canceled by the TSG client or the call is canceled because a shutdown sequence is initiated.

3.1.4.1.4 TsProxyCreateChannel (Opnum 4)

The **TsProxyCreateChannel** method is used to create a channel between the TSG client and the TSG server.<34> The TSG server SHOULD connect to the target server during this call to start communication between the TSG client and target server. If connection to the target server cannot be done, the TSG server MUST return HRESULT_CODE(E_PROXY_TS_CONNECTFAILED) as noted in the Return Values section.<35> The TSG server MUST return a representation of the channel to the TSG client. After this method call has successfully been completed, a channel shutdown can be performed by using the **TsProxyCloseChannel** method. Please refer to section 3.1.1 for a state transition diagram.

Prerequisites: The tunnel MUST be authorized; otherwise, the error ERROR_ACCESS_DENIED is returned.

Sequential Processing Rules:

1. If some unexpected error occurs during the following process, the TSG server MUST return E_PROXY_INTERNALERROR.
2. The TSG server MUST verify that the tunnel has been authorized. Otherwise, it MUST return ERROR_ACCESS_DENIED.
3. The TSG server MUST verify that the *tsEndPointInfo* parameter is not NULL and tsEndPointInfo->numResources is not zero. Otherwise, it MUST return ERROR_ACCESS_DENIED.
4. The TSG server MUST initialize the ADM element **Target server names** with combined array of the **resourceName** and **alternateResourceNames** members of the *tsEndPointInfo* parameter.
5. The TSG server MUST do the resource authorization as per policies configured at the TSG server. If the resource is not authorized, then it MUST return E_PROXY_RAP_ACCESSDENIED.<36>
6. If **Re-authentication Connection** is TRUE:
    1. The TSG server MUST find the original connection that has initiated the re-authentication using **Re-authentication Tunnel Context** and MUST set its ADM element **Re-authentication Status** to ResourceAuthorizationCompleted.
    2. Return ERROR_SUCCESS.
7. The TSG server SHOULD try to connect to the target server by each name in the target server names array until it succeeds or until the array is traversed completely. If connection fails for all

target server names, it MUST return HRESULT_CODE(E_PROXY_TS_CONNECTFAILED) in rpc_fault packet.

8. The TSG server MUST create the *channelId* and *channelContext* RPC content handles and MUST initialize the corresponding ADM elements.
9. The TSG server MUST also start the Session Timeout Timer (section 3.1.2.1), if the session timeout is configured at the TSG server.
10. If the TSG server is implementing the Connection Timer, the TSG server MUST start the Connection Timer.
11. The TSG server MUST return ERROR_SUCCESS.

```
HRESULT TsProxyCreateChannel(
  [in] PTUNNEL_CONTEXT_HANDLE_NOSERIALIZE tunnelContext,
  [in, ref] PTSENDPOINTINFO tsEndPointInfo,
  [out] PCHANNEL_CONTEXT_HANDLE_SERIALIZE* channelContext,
  [out] unsigned long* channelId
);
```

tunnelContext: The TSG client MUST provide the TSG server with the same context handle it received from the **TsProxyCreateTunnel** method call. The TSG server SHOULD throw an exception if the RPC validation and verification fails.

tsEndPointInfo: Pointer to the **TSENDPOINTINFO** structure. The TSG client MUST provide a non-NULL pointer to the TSG server for this structure. The TSG server initializes the ADM element Target server names with an array of **resourceName** and **alternateResourceNames** members of **TSENDPOINTINFO** structure. The TSG server SHOULD try to connect to the target server by each name in the array until it succeeds or until the array is traversed completely. If connection fails for all Target server names, HRESULT_CODE(E_PROXY_TS_CONNECTFAILED) (0x000059DD) is returned.<37> The rules for determining a valid server name are specified in section 2.2.1.1.

channelContext: A RPC context handle that represents context-specific information for the channel. The TSG server MUST provide a non-NULL value. The TSG client MUST save and use this context handle on all subsequent method calls on the channel. Specifically, these methods are **TsProxySendToServer**, **TsProxySetupReceivePipe**, and **TsProxyCloseChannel**.

channelId: An unsigned long identifying the channel. The TSG server MUST provide this value to the TSG client. The TSG client MUST save the returned channel ID for future use in the ADM element "Channel id" (section 3.2.1). This channel ID is not required on any future method calls.

Return Values: The method MUST return ERROR_SUCCESS on success. Other failures MUST be one of the codes listed. The client MAY interpret failures in any way it deems appropriate. See 2.2.2.24 for details on these errors.

Return value	State transition	Description
ERROR_SUCCESS (0x00000000)	The connection MUST transition to Channel	Returned when a call to the **TsProxyCreateChannel** method succeeds.

Return value	State transition	Description
	Created state.	
ERROR_ACCESS_DENIED (0x00000005)	The connection MUST NOT transition its state.	Returned either if *tunnelContext* parameter is NULL, if this method is called on a tunnel which is not authorized, if the **tsEndPointInfo** parameter is NULL, or if the **numResourceNames** member of the **tsEndPointInfo** parameter is zero.
E_PROXY_RAP_ACCESSDENIED (0x800759DA)	The connection MUST NOT transition its state.	Returned when an attempt to resolve or access a target server is blocked by TSG server policies.
E_PROXY_INTERNALERROR (0x800759D8)	The connection MUST NOT transition its state.	Returned when the server encounters an unexpected error while creating the channel.
HRESULT_CODE(E_PROXY_TS_CONNECTFAILED) (0x000059DD)	The connection MUST NOT transition its state.	This error is returned in rpc_fault packet when the TSG server fails to connect to any of the target server names, as specified in the members of **tsEndPointInfo**.

The error ERROR_ACCESS_DENIED is returned when this call is made on a tunnel which is not authorized.

3.1.4.2 Data Transfer Phase

3.1.4.2.1 TsProxySendToServer (Opnum 9)

The method is used for data transfer from the TSG client to the target server, via the TSG server. The TSG server SHOULD send the buffer data received in this method to the target server. The RPC runtime MUST NOT perform a strict NDR data consistency check for this method. The Terminal Services Gateway Server Protocol bypasses NDR for this method. The wire data MUST follow the regular RPC specifications as specified in [C706] section 2.1, and [MS-RPCE] minus all NDR headers, trailers, and NDR-specific payload. The TSG server MUST have created the channel to the target server before completing this method call. This method MAY be called multiple times by the TSG client, but only after the previous method call finishes. The TSG server MUST handle multiple sequential invocations of this method call. This method bypasses NDR. For this reason, unlike other RPC methods that return an HRESULT, this method returns a DWORD. This is directly passed to the callee from underlying RPC calls.<38> When this call fails, the TSG server MUST send the final response to **TsProxySetupReceivePipe** call.

Prerequisites: The connection MUST be in Pipe Created state. If this call is made in any other state, ERROR_ONLY_IF_CONNECTED is returned.

Sequential Processing Rules:

1. If some unexpected error occurs in the following process, the TSG server MUST return HRESULT_CODE(E_PROXY_INTERNALERROR).

2. The TSG server MUST extract the channel context handle from the *pRpcMessage* parameter. Refer to Generic Send Data Message Packet for the *pRpcMessage* format.

3. The TSG server MUST verify that the channel context handle is not NULL. Otherwise, it MUST return ERROR_ACCESS_DENIED.

4. The TSG server MUST verify that the connection is in Pipe Created state. Otherwise, it MUST return ERROR_ONLY_IF_CONNECTED.

5. The TSG server MUST extract the TSG client data from the *pRpcMessage* parameter. For the *pRpcMessage* format, refer to Generic Send Data Message Packet (section 2.2.3.3).

    1. The TSG server MUST verify that the **totalDataBytes** field in *pRpcMessage* is not zero. Otherwise, it MUST return ERROR_ACCESS_DENIED.

    2. The TSG server MUST verify that the **numBuffers** filed in *pRpcMessage* is in the range of 1 and 3, both inclusive. Otherwise, it MUST return ERROR_ACCESS_DENIED.

    3. The TSG server MUST verify that **buffer1Length** + **buffer2Length**, (if **numBuffers** >= 2), + **buffer3Length**, (if **numBuffers** == 3), + size of **buffer1Length** + size of **buffer2Length**, (if **numBuffers** >= 2), + size of **buffer3Length**, (if **numBuffers** == 3), does not exceed **totalDataBytes**. Otherwise, it MUST return ERROR_ACCESS_DENIED.

    4. The TSG server MUST verify that the **buffer1Length** field in *pRpcMessage* is not zero. Otherwise, it MUST return HRESULT_CODE(E_PROXY_INTERNALERROR).

    5. If **numBuffers** >= 2, then the TSG server MUST verify that the **buffer2Length** field is not zero. Otherwise, it MUST return HRESULT_CODE(E_PROXY_INTERNALERROR).

    6. If **numBuffers** == 3, then the TSG server MUST verify that the **buffer3Length** field is not zero. Otherwise, it MUST return HRESULT_CODE(E_PROXY_INTERNALERROR).

6. The TSG server MUST send the data extracted in the preceding step to the target server.

7. The TSG server MUST return ERROR_SUCCESS.

```
DWORD TsProxySendToServer(
  [in, max_is(32767)] byte pRpcMessage[]
);
```

pRpcMessage: The protocol data between TSG client and TSG server MUST be decoded as specified in section 2.2.3.3. RPC stub information is specified in [MS-RPCE] sections 1.1 and 1.5.

Return Values: The method MUST return ERROR_SUCCESS on success. Other failures MUST be one of the codes listed. The client MAY interpret failures in any way it deems appropriate. See 2.2.2.24 for details on these errors.

Return value	State transition	Description
ERROR_SUCCESS (0x00000000)	The connection MUST remain in PipeCreated state.	Returned when a call to the **TsProxySendToServer** method succeeds.
ERROR_ONLY_IF_CONNECTED (0x000004E3)	The connection MUST transition to Channel Close Pending state.	Returned by the TSG server when an attempt is made by the client to send data to the target server on connection state other than Pipe Created state. The TSG client MUST end the protocol when this error is returned.
ERROR_ACCESS_DENIED (0x00000005)	The connection MUST transition to Channel Close Pending state.	Returned if the channel context handle passed in the *pRpcMessage* parameter is NULL. The TSG client MUST end the protocol when this error is received.
HRESULT_CODE(E_PROXY_INTERNALERROR) (0x000059D8)	The connection MUST transition to Channel Close Pending state.	Returned when an unexpected error occurs in **TsProxySendToServer**. The TSG client MUST end the protocol when this error is received.

3.1.4.2.2 TsProxySetupReceivePipe (Opnum 8)

The **TsProxySetupReceivePipe** method is used for data transfer from the TSG server to the TSG client. The TSG server MUST create an RPC out pipe upon receiving this method call from the TSG client. This call bypasses the NDR and hence, the RPC runtime MUST NOT perform a strict NDR data consistency check for this method. Refer to section 3.3 for details on NDR-bypassing. Section 3.3.4 and section 3.3.5 give details on wire representation of data for responses to **TsProxySetupReceivePipe**. The out pipe MUST be created by the TSG server in the same manner as NDR creates it for a call.<39> The TSG server MUST use this out pipe and Stub Data field in RPC response PDUs to send all data from the target server to the TSG client on the channel. The TSG client MUST use this out pipe to pull data from the target server on the channel. On connection disconnect, the TSG server MUST send the following on the pipe: A **DWORD** return code in an RPC response PDU and set the PFC_LAST_FRAG bit in the **pfc_flags** field of the RPC response PDU. The pipe close is indicated when the PFC_LAST_FRAG bit is set in the **pfc_flags** field of the RPC response PDU. When the TSG client sees that the PFC_LAST_FRAG bit is set in the **pfc_flags** field of the RPC response PDU, it MUST interpret the 4 bytes Stub Data as the return code of **TsProxySetupReceivePipe**. For a description of RPC response PDU, **pfc_flags**, PFC_LAST_FRAG, and Stub Data, refer to sections 12.6.2and 12.6.4.10 in [C706]. The TSG client and TSG server MUST negotiate a separate out pipe for each channel. Out pipes MUST NOT be used or shared across channels.<40>

As long as the channel is not closed, the RPC and Transport layer guarantee that any data that is sent by the TSG server reaches the TSG client. RPC and Transport layer also ensure that the data is delivered to the TSG client in the order it was sent by the TSG server.

After the call reaches the TSG server, the connection MUST transition to Pipe Created state after setting up the out pipe.

Prerequisites: The connection MUST be in Channel Created state. If this is called in Pipe Created state or Channel Close Pending state, then the behavior is undefined. If this is called in any other state, ERROR_ACCESS_DENIED MUST be returned by the TSG server.

Sequential Processing Rules:

1. If some unexpected error occurs in the following process, the TSG server MUST return HRESULT_CODE(E_PROXY_INTERNALERROR).
2. If the TSG server is implementing the Connection Timer, then if **TsProxySetupReceivePipe** is called after the Connection Timer has expired, the TSG server MUST return ERROR_OPERATION_ABORTED; otherwise, the Connection Timer MUST be stopped.
3. The TSG server MUST extract the channel context handle from *pRpcMessage* parameter. For the *pRpcMessage* format, refer to TSG Client to TSG Server Packet Format (section 2.2.3.4.1).
4. The TSG server MUST verify that the channel context handle is not NULL. Otherwise, it MUST return ERROR_ACCESS_DENIED.
5. If the TSG server is configured such that the connections are allowed only to a resource that allows policy exchanges between the TSG server and the target server, and the target server does not support the same, then the TSG server MUST return HRESULT_CODE(E_PROXY_SDR_NOT_SUPPORTED_BY_TS).
6. If connection to the target server is not set up in **TsProxyCreateChannel** call, then the TSG server MUST try to connect to the target server by each name in the **Target server names** array until it succeeds or until the array is traversed completely. If connection fails for all target server names, it MUST return HRESULT_CODE(E_PROXY_TS_CONNECTFAILED).<41>
7. The TSG server MUST set up an out pipe to send data received from the target server to the TSG client.
8. The connection MUST transition to Pipe Created state.
9. The TSG server MUST start receiving data from the target server and stream the same to the TSG client. This process MUST be continued until one of the following events occurs.
    1. If the Session Timeout Timer expires and "disconnect on session timeout" is configured at the TSG server:
        1. If the Abstract Data Model (ADM) element **Negotiated Capabilities** contains **TSG_NAP_CAPABILITY_IDLE_TIMEOUT**, then the TSG server MUST disconnect the session by sending the final response of the **TsProxySetupReceivePipe** method with the HRESULT_CODE(E_PROXY_SESSIONTIMEOUT) error code.
        2. If the ADM element **Negotiated Capabilities** does not contain **TSG_NAP_CAPABILITY_IDLE_TIMEOUT**, then the TSG server MUST disconnect the session by sending the final response of the **TsProxySetupReceivePipe** method with the HRESULT_CODE(E_PROXY_CONNECTIONABORTED) error code.
    2. If the session timeout timer expires and "re-authentication on session timeout" is configured at the TSG server, the TSG server initiates a re-authentication with the client and starts the re-authentication timer, as explained in section 3.1.5.1. After the re-authentication timer expires, the TSG server MUST check the value of **Re-authentication Status** ADM element.

- If the ADM element **Re-authentication Status** is set to NONE:
  1. If the ADM element **Negotiated Capabilities** contains **TSG_NAP_CAPABILITY_IDLE_TIMEOUT**, then the TSG server MUST disconnect the connection with HRESULT_CODE(E_PROXY_REAUTH_AUTHN_FAILED).
  2. If the ADM element **Negotiated Capabilities** does not contain **TSG_NAP_CAPABILITY_IDLE_TIMEOUT**, then the TSG server MUST disconnect the connection with HRESULT_CODE(E_PROXY_CONNECTIONABORTED).
- If the ADM element **Re-authentication Status** is set to AuthenticationCompleted:
  1. If the ADM element **Negotiated Capabilities** contains **TSG_NAP_CAPABILITY_IDLE_TIMEOUT**, then the TSG server MUST disconnect the connection with HRESULT_CODE(E_PROXY_REAUTH_CAP_FAILED).
  2. If the ADM element **Negotiated Capabilities** does not contain **TSG_NAP_CAPABILITY_IDLE_TIMEOUT**, then the TSG server MUST disconnect the connection with HRESULT_CODE(E_PROXY_CONNECTIONABORTED).
- If the ADM element **Re-authentication Status** is set to UserAuthorizationCompletedButQurantineFailed:
  1. If the ADM element **Negotiated Capabilities** contains **TSG_NAP_CAPABILITY_IDLE_TIMEOUT**, then the TSG server MUST disconnect the connection with HRESULT_CODE(E_PROXY_REAUTH_NAP_FAILED).
  2. If the ADM element **Negotiated Capabilities** does not contain **TSG_NAP_CAPABILITY_IDLE_TIMEOUT**, then the TSG server MUST disconnect the connection with HRESULT_CODE(E_PROXY_CONNECTIONABORTED).
- If the ADM element **Re-authentication Status** is set to UserAuthorizationCompleted:
  1. If the ADM element **Negotiated Capabilities** contains **TSG_NAP_CAPABILITY_IDLE_TIMEOUT**, then the TSG server MUST disconnect the connection with HRESULT_CODE(E_PROXY_REAUTH_RAP_FAILED).
  2. If the ADM element **Negotiated Capabilities** does not contain **TSG_NAP_CAPABILITY_IDLE_TIMEOUT**, then the TSG server MUST disconnect the connection with HRESULT_CODE(E_PROXY_CONNECTIONABORTED).
- If the ADM element **Re-authentication Status** is set to ResourceAuthorizationCompleted, the TSG server MUST start the Session Timeout Timer and MUST reset the ADM element **Re-authentication Status** to NONE.

3. If the target server unexpectedly closes the connection between the TSG server and the target server, the TSG server MUST return ERROR_BAD_ARGUMENTS.
4. If the TSG server administrator forcefully disconnects the connection, the TSG server MUST return HRESULT_CODE(E_PROXY_CONNECTIONABORTED).
5. If the connection gets disconnected either by the TSG client or the TSG server, or by an unknown error, the TSG server MUST send the corresponding error code to the TSG client in the final response, as specified in TSG Server to TSG Client Packet Format for Final Response (section 2.2.3.4.3).

```
DWORD TsProxySetupReceivePipe(
   [in, max_is(32767)] byte pRpcMessage[]
);
```

pRpcMessage: The protocol data between TSG client and TSG server MUST be decoded as specified in section 2.2.3.4. RPC stub information is specified in [MS-RPCE] sections 1.1 and 1.5.

Return Values: The method MUST return ERROR_GRACEFUL_DISCONNECT on success, that is, if the TSG client gracefully disconnects the connection by calling **TsProxyCloseChannel**. Other failures MUST be one of the codes listed. The client MAY interpret failures in any way it deems appropriate. See 2.2.2.24 for details on these errors.

The error DWORD value is always sent, when the receive pipe closes down. The receive pipe will always close down when a disconnect takes place.

Return value	State transition	Description
ERROR_ACCESS_DENIED (0x00000005)	The connection MUST transition to Tunnel Close Pending state.	Returned either if this method is called before **TsProxyCreateChannel** or if the **Channel Context Handle** ADM element is NULL. The TSG client MUST end the protocol when this error is received.
HRESULT_CODE(E_PROXY_INTERNALERROR) (0x000059D8)	The connection MUST transition to Tunnel Close Pending state.	Returned when an unexpected error occurs in **TsProxySetupReceivePipe**. The TSG client MUST end the protocol when this error is received.
HRESULT_CODE(E_PROXY_TS_CONNECTFAILED) (0x000059DD)	The connection MUST transition to Tunnel Close Pending state.	Returned when the TSG server fails to connect to target server. It is returned in an rpc_fault packet.<42> The TSG client MUST end the protocol when this error is received.
HRESULT_CODE(E_PROXY_SESSIONTIMEOUT) (0x000059F6)	The connection MUST transitio	Returned by TSG server if a session timeout occurs and "disconnect on session timeout" is configured at the TSG server and the ADM element **Negotiated Capabilities** contains

Return value	State transition	Description
	n to Tunnel Close Pending state.	**TSG_NAP_CAPABILITY_IDLE_TIMEOUT**. The TSG client MUST end the protocol when this error is received.
HRESULT_CODE(E_PROXY_REAUTH_AUTHN_FAILED) (0x000059FA)	The connection MUST transition to Tunnel Close Pending state.	Returned when a re-authentication attempt by the client has failed because the user credentials are no longer valid and the ADM element **Negotiated Capabilities** contains **TSG_NAP_CAPABILITY_IDLE_TIMEOUT**. The TSG client MUST end the protocol when this error is received.
HRESULT_CODE(E_PROXY_REAUTH_CAP_FAILED) (0x000059FB)	The connection MUST transition to Tunnel Close Pending state.	Returned when a re-authentication attempt by the client has failed because the user is not authorized to connect through the TSG server anymore and the ADM element **Negotiated Capabilities** contains **TSG_NAP_CAPABILITY_IDLE_TIMEOUT**. The TSG client MUST end the protocol when this error is received.
HRESULT_CODE(E_PROXY_REAUTH_RAP_FAILED) (0x000059FC)	The connection MUST transition to Tunnel Close Pending state.	Returned when a re-authentication attempt by the client has failed because the user is not authorized to connect to the given end resource anymore and the ADM element **Negotiated Capabilities** contains **TSG_NAP_CAPABILITY_IDLE_TIMEOUT**. The TSG client MUST end the protocol when this error is received.
HRESULT_CODE(E_PROXY_CONNECTIONABORTED) (0x000004D4)	The connection MUST transition to Tunnel Close Pending state.	Returned when the following happens: 1. The TSG server administrator forcefully disconnects the connection. 2. Or when the ADM element **Negotiated Capabilities** doesn't contain **TSG_NAP_CAPABILITY_IDLE_TIMEOUT** and any one of the following happens: 1. Session timeout occurs and disconnect on session timeout is configured at the

Return value	State transition	Description
		TSG server. 2. Re-authentication attempt by the client has failed because the user credentials are no longer valid. 3. Re-authentication attempt by the client has failed because the user is not authorized to connect through the TSG server anymore. 4. Re-authentication attempt by the client has failed because the user is not authorized to connect to the given end resource anymore. 5. Re-authentication attempt by the TSG client has failed because the health of the user's computer is no longer compliant with the TSG server configuration. The TSG client MUST end the protocol when this error is received.
HRESULT_CODE(E_PROXY_SDR_NOT_SUPPORTED_BY_TS) (0x000059FD)	The connection MUST transition to Tunnel Close Pending state.	The TSG server is capable of exchanging policies with some target servers. The TSG server MAY be configured to allow connections to only target servers that are capable of policy exchange. If such a setting is configured and the target server is not capable of exchanging policies with the TSG server, this error will be returned. The TSG client MUST end the protocol when this error is received.
ERROR_GRACEFUL_DISCONNECT (0x000004CA)	The connection MUST transition to Tunnel Close Pending state.	Returned when the connection is disconnected gracefully by the TSG client calling **TsProxyCloseChannel**.
HRESULT_CODE(E_PROXY_REAUTH_NAP_FAILED) (0x00005A00)	The connecti	Returned when a re-authentication attempt by the TSG client has failed

Return value	State transition	Description
	on MUST transition to Tunnel Close Pending state.	because the user's computer's health is no longer compliant with the TSG server configuration and the ADM element **Negotiated Capabilities** contains **TSG_NAP_CAPABILITY_IDLE_TIMEOUT**. The TSG client MUST end the protocol when this error is received.
ERROR_OPERATION_ABORTED(0x000003E3)	The connection MUST transition to Tunnel Close Pending state.	Returned when the call to **TsProxySetupReceivePipe** is received after the Connection Timer has expired.
ERROR_BAD_ARGUMENTS(0x000000A0)	The connection MUST transition to Tunnel Close Pending state.	Returned when the target server unexpectedly closes the connection between the TSG server and the target server.

3.1.4.3 Shutdown Phase

Shutdown phase is used to terminate the channel and tunnel. Channel closure can either be initiated by the TSG client or the TSG server. The TSG client SHOULD initiate it by closing the channel using method TsProxyCloseChannel. The TSG server initiates it by setting the PFC_LAST_FRAG bit in the **pfc_flags** field in the final response for the TsProxySetupReceivePipe method. If the client has any pending administrative message requests on the TSG server, the client cancels the same by making a TsProxyMakeTunnelCall call with TSG_TUNNEL_CANCEL_ASYNC_MSG_REQUEST as a parameter. The closing of tunnel is accomplished by using the TsProxyCloseTunnel method.

3.1.4.3.1 TsProxyCloseChannel (Opnum 6)

The **TsProxyCloseChannel** method is used to terminate the channel from the TSG client to the TSG server. This SHOULD be called only if the TSG client has not received the RPC response PDU with the PFC_LAST_FRAG bit set in the **pfc_flags** field. All communication between the TSG client and the target server MUST stop after the TSG server executes this method. The TSG client MUST NOT use this context handle in any subsequent operations after calling this method. This will terminate the channel between the TSG client and the target server. If the TSG server has not already sent the RPC response PDU with the PFC_LAST_FRAG bit set in the **pfc_flags** field, which happens if the TSG server initiated the disconnect, the TSG client will also receive a return code for TsProxySetupReceivePipe in an RPC response PDU with the PFC_LAST_FRAG bit set in the

pfc_flags. For a description of RPC response PDU, pfc_flags, and PFC_LAST_FRAG, refer to sections 12.6.2 and 12.6.14.10 in [C706].

The TSG server completes the **TsProxyCloseChannel** only after sending all of the data it received before this call was made. The TSG client receives the call complete notification only after it receives all of the data that was sent by the TSG server before completing **TsProxyCloseChannel**. Please refer to section 3.1.4.2.2 for details on how the data is ensured to reach the destination.

Prerequisites: The connection MUST be in Channel Created state or Pipe Created state or Channel Close Pending state.

Sequential Processing Rules:

1. The TSG server MUST check whether the channel context handle is NULL or not a valid context handle. If so, the TSGU server MUST return ERROR_ACCESS_DENIED.

2. The TSG server MUST disconnect the connection to the target server.

3. The TSG server MUST send all data received from the target server to the TSG client and MUST end **TsProxySetupReceivePipe** with ERROR_GRACEFUL_DISCONNECT.

4. The TSG server MUST return ERROR_SUCCESS.

```
HRESULT TsProxyCloseChannel(
  [in, out] PCHANNEL_CONTEXT_HANDLE_NOSERIALIZE* context
);
```

context: The TSG client MUST provide the TSG server with the same context handle it received from the **TsProxyCreateChannel** method call.

Return Values:

Return value	State transition	Description
ERROR_SUCCESS (0x00000000)	The connection MUST transition to Tunnel Close Pending state.	Returned when the call to the **TsProxyCloseChannel** method succeeds.
ERROR_ACCESS_DENIED (0x00000005)	The connection MUST NOT transition its state.	Returned when the provided context parameter is NULL or not a valid channel context handle.

3.1.4.3.2 TsProxyMakeTunnelCall (Opnum 3)

The **TsProxyMakeTunnelCall** method MUST be called with the TSG_TUNNEL_CANCEL_ASYNC_MSG_REQUEST parameter before the **TsProxyCloseTunnel** method is called if the previous **TsProxyMakeTunnelCall** has not returned. The **TsProxyMakeTunnelCall** method has been defined in section 3.1.4.1.3.

The **TsProxyCloseTunnel** method call uses a serialized context handle. If a previous call to the **TsProxyMakeTunnelCall** has not returned, then the TSG client cannot call **TsProxyCloseTunnel**, because of the serialized nature of the context handle.

3.1.4.3.3 TsProxyCloseTunnel (Opnum 7)

The **TsProxyCloseTunnel** method is used to terminate the tunnel between the TSG client and the TSG server. All communication between the TSG client and TSG server MUST stop after the TSG server executes this method. The TSG client MUST NOT use this tunnel context handle in any subsequent operations after this method call. This MUST be the final tear down phase of the TSG client to TSG server tunnel. If the ADM element **Re-authentication Connection** is FALSE, then the ADM element **Number of Connections** MUST be decremented by 1 in this call. If there is an existing channel within the tunnel, it SHOULD first be closed using **TsProxyCloseChannel**. If the TSG client calls the **TsProxyCloseTunnel** method before calling the **TsProxyCloseChannel** method, the TSG server MUST close the channel and then close the tunnel.

Prerequisites: The connection MUST be in any of the following states: Connected state, Authorized state, Channel Created state, Pipe Created state, Channel Close Pending state, or Tunnel Close Pending state.

Sequential Processing Rules:

1. The TSG server MUST check whether the tunnel context handle is NULL or not a valid context handle. If so, it MUST return ERROR_ACCESS_DENIED.

2. If there are any channels in the tunnel then the TSG server MUST disconnect them. If **TsProxyCloseChannel** has not already been called then the TSG server MUST close the RPC out pipe and return ERROR_GRACEFUL_DISCONNECT for the **TsProxySetupReceivePipe**.

3. The TSG server MUST disconnect the tunnel.

4. If the ADM element **Re-authentication Connection** is FALSE:

   - The TSG server MUST decrement the ADM element **Number of Connections** by 1.

5. The connection MUST transition to the End state.

6. The TSG server MUST return ERROR_SUCCESS.

```
HRESULT TsProxyCloseTunnel(
  [in, out] PTUNNEL_CONTEXT_HANDLE_SERIALIZE* context
);
```

context: The TSG client MUST provide the TSG server with the same context handle it received from the **TsProxyCreateTunnel** method call.

Return Values: The method MUST return 0 on success. This function SHOULD NOT fail from a TSG protocol perspective. If **TsProxyCloseTunnel** is called while any of the channels are not closed, then the TSG server MUST close all the channels and then close the tunnel.

Return value	State transition	Description
ERROR_SUCCESS (0x00000000)	The connection MUST transition to the end state.	Returned when a call to the **TsProxyCloseTunnel** method succeeds.
ERROR_ACCESS_DENIED (0x00000005)	The connection MUST NOT transition its state.	Returned when the provided context parameter is NULL or not a valid tunnel context handle.

3.1.4.3.4 Server Initiated Shutdown

The server initiates shutdown by sending the final response packet to TsProxySetupReceivePipe call with the PFC_LAST_FRAG bit set in the **pfc_flags** field. The server closes the channel after sending this response. The client SHOULD not call TsProxyCloseChannel after receiving this final response. The client SHOULD call the TsProxyCloseChannel method if the client initiates the shutdown, but not if the server initiates shutdown.

Prerequisites: The connection MUST be in Pipe Created state.

Sequential Processing Rules:

1. The TSG server MUST send the final response packet to TsProxySetupReceivePipe call with the PFC_LAST_FRAG bit set in the **pfc_flags** field.
2. The TSG server MUST close the channel.
3. The connection MUST be transitioned to Tunnel Close Pending state.

3.1.5 Timer Events

3.1.5.1 Session Timeout Timer

1. If the Session Timeout Timer expires and "disconnect on session timeout" is configured at the TSG server, then review the following.
   1. If the ADM element **Negotiated Capabilities** contains **TSG_NAP_CAPABILITY_IDLE_TIMEOUT**, then the TSG server MUST disconnect the session by sending the final response of the **TsProxySetupReceivePipe** method with the HRESULT_CODE(E_PROXY_SESSIONTIMEOUT) error code.
   2. If the ADM element **Negotiated Capabilities** doesn't contain **TSG_NAP_CAPABILITY_IDLE_TIMEOUT**, then the TSG server MUST disconnect the session by sending the final response of the **TsProxySetupReceivePipe** method with the HRESULT_CODE(E_PROXY CONNECTIONABORTED) error code.
2. Otherwise, if this timer expires and "re-authentication on session timeout" is configured at the TSG server, the TSG server MUST initiate the re-authentication connection as follows:
   1. The TSG server MUST set the ADM element **Re-authentication Status** to None.
   2. The TSG server MUST start the Re-authentication Timer.
   3. If there is no waiting **TsProxyMakeTunnelCall** call, do nothing.
   4. If there is a waiting **TsProxyMakeTunnelCall** call:
      1. The TSG server MUST set the **packetId** member of the *tsgPacketResponse* out parameter of **TsProxyMakeTunnelCall** to **TSG_PACKET_TYPE_MESSAGE_PACKET**.
      2. The TSG server MUST set tsgPacketResponse->packetMsgResponse->msgType to **TSG_ASYNC_MESSAGE_REAUTH**.
      3. The TSG server MUST initialize tsgPacketResponse->packetMsgResponse->messagePacket.reauthMessage->tunnelContext by the ADM element **Re-authentication Tunnel Context**.

4. The TSG server MUST complete the waiting **TsProxyMakeTunnelCall** with error code ERROR_SUCCESS.

3.1.5.2 Re-authentication Timer

If the Re-authentication Timer expires, the TSG server MUST check the ADM element **Re-authentication Status** value.

- If the ADM element **Re-authentication Status** is set to NONE:
  1. If the ADM element **Negotiated Capabilities** contains **TSG_NAP_CAPABILITY_IDLE_TIMEOUT**, then the TSG server MUST disconnect the connection with HRESULT_CODE(E_PROXY_REAUTH_AUTHN_FAILED).
  2. If the ADM element **Negotiated Capabilities** doesn't contain **TSG_NAP_CAPABILITY_IDLE_TIMEOUT**, then the TSG server MUST disconnect the connection with HRESULT_CODE(E_PROXY_CONNECTIONABORTED).
- If the ADM element **Re-authentication Status** is set to AuthenticationCompleted:
  1. If the ADM element **Negotiated Capabilities** contains **TSG_NAP_CAPABILITY_IDLE_TIMEOUT**, then the TSG server MUST disconnect the connection with HRESULT_CODE(E_PROXY_REAUTH_CAP_FAILED).
  2. If the ADM element **Negotiated Capabilities** doesn't contain **TSG_NAP_CAPABILITY_IDLE_TIMEOUT**, then the TSG server MUST disconnect the connection with HRESULT_CODE(E_PROXY_CONNECTIONABORTED).
- If the ADM element **Re-authentication Status** is set to UserAuthorizationCompletedButQurantineFailed:
  1. If the ADM element **Negotiated Capabilities** contains **TSG_NAP_CAPABILITY_IDLE_TIMEOUT**, then the TSG server MUST disconnect the connection with HRESULT_CODE(E_PROXY_REAUTH_NAP_FAILED).
  2. If the ADM element **Negotiated Capabilities** doesn't contain **TSG_NAP_CAPABILITY_IDLE_TIMEOUT**, then the TSG server MUST disconnect the connection with HRESULT_CODE(E_PROXY_CONNECTIONABORTED).
- If the ADM element **Re-authentication Status** is set to UserAuthorizationCompleted:
  1. If the ADM element **Negotiated Capabilities** contains **TSG_NAP_CAPABILITY_IDLE_TIMEOUT**, then the TSG server MUST disconnect the connection with HRESULT_CODE(E_PROXY_REAUTH_RAP_FAILED).
  2. If the ADM element **Negotiated Capabilities** doesn't contain **TSG_NAP_CAPABILITY_IDLE_TIMEOUT**, then the TSG server MUST disconnect the connection with HRESULT_CODE(E_PROXY_CONNECTIONABORTED).
- If the ADM element **Re-authentication Status** is set to ResourceAuthorizationCompleted, the TSG server MUST start the Session Timeout Timer and MUST reset the ADM element **Re-authentication Status** to NONE.

3.1.5.3 Connection Timer

If the Connection Timer expires and the call to the **TsProxySetupReceivePipe** method is received by the TSG server after the timer has expired, the server MUST disconnect with the ERROR_OPERATION_ABORTED return value, as specified in section 2.2.2.24.

3.1.6 Other Local Events

This section describes the use of an abstract interface on the server between NAPSO and the TSGU server. This interface is not used between the TSGU client and the TSGU server.

SoHRAsyncCallback

When NAPSO finishes processing the statement of health (SoH) on the TSG server, it sends the statement of health response (SoHR) to the TSG server, as specified in [MS-NAPSO] section 10.3.7, in the form of SoHR, as described in [MS-SOH], using this abstract interface.

- **Inputs**: None.
- **Outputs**:
  - *dwSoHRSize*: A 32-bit unsigned integer specifying the number of bytes returned in the *ppSoHR* parameter.
  - *ppSoHR*: Pointer to buffers returning the SoHR.
  - *dwResponse*: A 32-bit unsigned integer specifying the type of response.
  - *hr*: HRESULT specifying result of the SoH processing by NAPSO.
- **Constraints**:
  - The TSG server MUST not allow the connection if the value of the *dwResponse* indicates that the TSG client's health is not acceptable.

3.1.6.1 Data Arrival from Target Server

This event occurs when the target server data arrives at the TSG server, which is destined for the TSG client. When this event occurs, the TSG server MUST stream the data to the TSG client, in response to the **TsProxySetupReceivePipe**, in the same order that it arrived.

3.2 TsProxyRpcInterface Client Details

The following sections contain the details of the TsProxyRpcInterface on the Client.

3.2.1 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 that described in this document.

Target server name: A string of Unicode characters. The server name applies to the machine that the TSG server connects to.<43>

Client Machine name: A string of Unicode characters that cannot exceed 513 bytes,<44> including the terminating null character. The Client Machine name refers to the machine that runs the TSG client. It is possible for the Client Machine name to be the same as the server name (in value) if the client and the server run on the same physical machine.<45>

Tunnel id: An unsigned long representing the tunnel identifier for tracking purposes on the TSG server. It MAY be used by the TSG client to help the TSG server administrator troubleshoot connection issues.

Channel id: An unsigned long representing the channel identifier for tracking purposes on the TSG server. It MAY be used by the TSG client to help the TSG server administrator troubleshoot connection issues.

Binding Handle: An RPC binding handle created by the TSG client to bind to the TSG server. For more details about Binding Handle, see [C706] section 2.1.

Tunnel Context handle: An RPC context handle for the TSG client to TSG server represented by an array of 20 bytes on the TSG server. The context handle is used to identify a specific connection from the TSG client to the TSG server.

Channel Context handle: An RPC context handle for the connection from the TSG client to the target server via the TSG server represented by an array of 20 bytes on the TSG server. The context handle is used to identify a specific connection to the target server from the TSG client via the TSG server.

CertChainData: A string of variable data returned by the TSG server representing the certificate chain used by the TSG server for the HTTPS communication between TSG client and TSG server. The TSG client MAY use this data to verify the identity of the TSG server before sending sensitive data, such as the health information of the TSG client machine.

Nonce: A unique GUID returned by the TSG server to identify the current connection. The TSG client sends this GUID to the TSG server if it sends the statement of health (SoH), as specified in section 2.2.3.2.1.4.

Idle Timeout Value: An unsigned long value that specifies connection idle time in minutes before the connection is torn down.

Negotiated Capabilities: A ULONG bitmask value representing the negotiated capabilities between the TSG client and the TSG server. It contains zero or more of the following values:

NAP Capability Value
TSG_NAP_CAPABILITY_QUAR_SOH
TSG_NAP_CAPABILITY_IDLE_TIMEOUT
TSG_MESSAGING_CAP_CONSENT_SIGN
TSG_MESSAGING_CAP_SERVICE_MSG
TSG_MESSAGING_CAP_REAUTH

3.2.2 Timers

3.2.2.1 Idle Timeout Timer

If idle timeout capability is negotiated between the TSG client and the TSG server, then the TSG server MUST send the idle timeout value to the TSG client in the **TSG_PACKET_RESPONSE** structure in response to the **TsProxyAuthorizeTunnel** call. If idle timeout is not configured at the TSG server, it MUST send zero.

3.2.2.1.1 Idle Time Processing

If the idle timeout value is zero, no idle timeout is configured at the TSG server, and therefore, no idle time processing is required by the TSG client.

If the idle timeout value is nonzero, the TSG client SHOULD start this timer and SHOULD reset the timer whenever the TSG client sends some payload data in the **TsProxySendToServer (section 3.1.4.2.1)** method to the TSG server. The TSG client SHOULD end the protocol when the timer expires as the connection has been idle for the specified Idle Timeout Value.

Other than that described in this section, no protocol timers are required beyond those used internally by RPC to implement resiliency to network outages, as specified in [MS-RPCE] section 3.

3.2.3 Initialization

The TSG client creates an RPC binding handle to the TSG server's RPC endpoint. The TSG client MUST create a binding handle, a binding handle is specified in [C706] section 2.1, and make the first method invocation to receive the tunnel context handle, as specified in section 3.1.4.1. Subsequent method invocations MUST use either the tunnel context handle or the channel context handle, as each method requires. The TSG client MUST create an authenticated RPC binding handle with a minimum of RPC_C_AUTHN_LEVEL_PKT_INTEGRITY and other parameters as specified in section 2.1. This requires establishing the binding to the well-known endpoint as specified in section 2.1.

If an authenticated binding handle is established, the TSG client MUST match the version and capabilities of the TSG server; if no match can be made, the TSG client SHOULD stop further progress on the protocol connection.

3.2.4 Message Processing Events and Sequencing Rules

This protocol asks the RPC runtime to perform a strict NDR data consistency check at target level 7.0 for all methods unless otherwise specified, as specified in [MS-RPCE] section 1.3.

All the methods implemented by the TSG server SHOULD enforce appropriate security measures to make sure that the TSG client has the required permissions to execute the routines. All methods MUST be RPC calls. However, these methods MUST be called in a sequence specified in section 1.3.

The methods MAY throw an exception and the TSG client MUST handle these exceptions appropriately. The methods called by the TSG client MUST be sequential in order, as specified in section 1.3.1. The method details are specified in section 3.1.4.

A TSG client's invocation of each method is typically the result of local application activity. The local application at the TSG client specifies values for all input parameters. No other higher-layer triggered events are processed.

The TSG client SHOULD process errors returned from the TSG server and notify the application invoker of the error received in the higher layer.

Sequential processing rules for connection process:

1. The TSG client MUST call **TsProxyCreateTunnel** to create a tunnel to the gateway.
2. If the call fails, the TSG client MUST end the protocol and MUST NOT perform the following steps.
3. The TSG client MUST initialize the following ADM elements using **TsProxyCreateTunnel** out parameters:
    1. The TSG client MUST initialize the ADM element **Tunnel id** with the *tunnelId* out parameter.
    2. The TSG client MUST initialize the ADM element **Tunnel Context Handle** with the *tunnelContext* out parameter. This **Tunnel Context Handle** is used for subsequent tunnel-related calls.
    3. If *tsgPacketResponse*->packetId is **TSG_PACKET_TYPE_CAPS_RESPONSE**, where *tsgPacketResponse* is an out parameter,
        1. The TSG client MUST initialize the ADM element **Nonce** with tsgPacketResponse->tsgPacket.packetCapsResponse->pktQuarEncResponse.nonce.
        2. The TSG client MUST initialize the ADM element **Negotiated Capabilities** with tsgPacketResponse->tsgPacket.packetCapsResponse->pktQuarEncResponse.versionCaps->tsgCaps[0].tsgPacket.tsgCapNap.capabilities.
    4. If *tsgPacketResponse*->packetId is **TSG_PACKET_TYPE_QUARENC_RESPONSE**, where *tsgPacketResponse* is an out parameter,
        1. The TSG client MUST initialize the ADM element **Nonce** with tsgPacketResponse->tsgPacket.packetQuarEncResponse->nonce.
        2. The TSG client MUST initialize the ADM element **Negotiated Capabilities** with tsgPacketResponse->tsgPacket.packetQuarEncResponse->versionCaps->tsgCaps[0].tsgPacket.tsgCapNap.capabilities.
4. The TSG client MUST get its statement of health (SoH) by calling **NAP EC API**, as specified in [MS-NAPSO] section 3.3.1.<46>Details of the SoH format are specified in [MS-SOH] section 2.2.5. If the SoH is received successfully, then the TSG client MUST encrypt the SoH using the **Triple Data Encryption Standard** algorithm and encode it with the TSG server certificate context available in the ADM element **CertChainData**.
5. The TSG client MUST copy the ADM element **Nonce** to tsgPacket.packetQuarRequest->data and append the encrypted SoH message into tsgPacket.packetQuarRequest->data. The TSG client MUST set the tsgPacket.packetQuarRequest->dataLen to the sum of the number of bytes in the encrypted SoH message and number of bytes in the ADM element **Nonce**, where *tsgpacket* is an input parameter of **TsProxyAuthorizeTunnel**. The format of the **packetQuarRequest** field is specified in section 2.2.3.2.1.4.
6. The TSG client MUST call **TsProxyAuthorizeTunnel** to authorize the tunnel.
7. If the call succeeds or fails with error E_PROXY_QUARANTINE_ACCESSDENIED, follow the steps later in this section. Else, the TSG client MUST end the protocol and MUST NOT follow the steps later in this section.
8. If the ADM element **Negotiated Capabilities** contains **TSG_NAP_CAPABILITY_IDLE_TIMEOUT**, then the ADM element **Idle Timeout Value** SHOULD be initialized with first 4 bytes of tsgPacketResponse->tsgPacket.packetResponse-

>responseData and the **Statement of health response** variable should be initialized with the remaining bytes of **responseData**, where *tsgPacketResponse* is an out parameter of **TsProxyAuthorizeTunnel**. The format of the **responseData** member is specified in section 2.2.3.2.1.5.1.

9. If the ADM element **Negotiated Capabilities** doesn't contain **TSG_NAP_CAPABILITY_IDLE_TIMEOUT**, then the ADM element **Idle Timeout Value** SHOULD be initialized to zero and the **Statement of health response** variable should be initialized with all the bytes of tsgPacketResponse->tsgPacket.packetResponse->responseData.

10. Verify the signature of the **Statement of health response** variable and decode it using the TSG server certificate context available in the ADM element **CertChainData**, and pass it to Process SoHR Task (section 14) as specified in [MS-NAPSO] section 14.

11. If the call **TsProxyAuthorizeTunnel** fails with error E_PROXY_QUARANTINE_ACCESSDENIED, the TSG client MUST end the protocol and MUST NOT follow the steps later in this section.

12. If the ADM element **Idle Timeout Value** is nonzero, the TSG client SHOULD start the idle time processing as specified in section 3.2.2.1.1 and SHOULD end the protocol when the connection has been idle for the specified **Idle Timeout Value**.

13. If the ADM element **Negotiated Capabilities** contains **TSG_MESSAGING_CAP_SERVICE_MSG**, a **TsProxyMakeTunnelCall** call MAY be made by the client, with **TSG_TUNNEL_CALL_ASYNC_MSG_REQUEST** as the parameter, to receive messages from the TSG server.

14. The TSG client MUST call **TsProxyCreateChannel** to create a channel to the target server name as specified by the ADM element **Target Server Name** (section 3.2.1).

15. If the call fails, the TSG client MUST end the protocol and MUST not follow the below steps.

16. The TSG client MUST initialize the following ADM elements using **TsProxyCreateChannel** out parameters.

    1. The TSG client MUST initialize the ADM element **Channel id** with the *channelId* out parameter.

    2. The TSG client MUST initialize the ADM element **Channel Context Handle** with the *channelContext* out parameter. This **Channel Context Handle** is used for subsequent channel-related calls.

Sequential processing rules for data transfer:

1. The TSG client MUST call **TsProxySetupReceivePipe** to receive data from the target server, via the TSG server.

2. The TSG client MUST call **TsProxySendToServer** to send data to the target server via the TSG server, and if the Idle Timeout Timer is started, the TSG client SHOULD reset the Idle Timeout Timer.

3. If **TsProxyMakeTunnelCall** is returned, the TSG client MUST process the message and MAY call **TsProxyMakeTunnelCall** again with **TSG_TUNNEL_CALL_ASYNC_MSG_REQUEST** as the parameter.

4. The TSG client MUST end the protocol after it receives the final response to **TsProxySetupReceivePipe**. The final response format is specified in section 2.2.3.4.3.

Sequential processing rules for ending the protocol:

1. If a channel was successfully created in the connection process, the TSG client MUST call **TsProxyCloseChannel** to close the channel.

2. If the TSG client called **TsProxyMakeTunnelCall** during the connection process and the call has not yet returned, the TSG client MUST call **TsProxyMakeTunnelCall** with the **TSG_TUNNEL_CANCEL_ASYNC_MSG_REQUEST** parameter to cancel the previous pending call.

3. If the tunnel was successfully created during the connection process, the TSG client MUST call **TsProxyCloseTunnel** to close the tunnel.

Sequential processing rules when the TSG client receives a re-authentication message from the TSG server:

1. The TSG client MUST start a new connection by calling **TsProxyCreateTunnel**. The **packetId** member of the *tsgPacket* MUST be set to **TSG_PACKET_TYPE_REAUTH**. Also, tsgPacket->packetReauth.tunnelContext MUST be initialized by the tsgPacketResponse->packetMsgResponse->messagePacket.reauthMessage->tunnelContext, which is received in the **TsProxyMakeTunnelCall** response.

2. If **TsProxyCreateTunnel** fails, go to step 6.

3. On successful completion of **TsProxyCreateTunnel**, the TSG client MUST call **TsProxyAuthorizeTunnel**.

4. If **TsProxyAuthorizeTunnel** fails, go to step 6.

5. On successful completion of **TsProxyAuthorizeTunnel**, the TSG client MUST call **TsProxyCreateChannel**.

6. End of processing re-authentication message.

Other than the above, no other special message processing is required on the TSG client beyond the processing required in the underlying RPC protocol, as specified in [MS-RPCE].

3.2.5 Timer Events

None.

3.2.5.1 Idle Timeout Timer

If the Idle Timeout Timer expires, the TSG client SHOULD end the protocol.

3.2.6 Other Local Events

Whenever there is a change in the TSG client computer's health, NAPSO informs the TSG client. However, because the TSG client does not process the change, this does not result in an exchange of data between the TSG client and the TSG server.

3.3 Data Representation forTsProxySetupReceivePipe and TsProxySendToServer

NDR64 specifies a method to package the data before sending it on the wire. For improved performance, **TsProxySetupReceivePipe** and **TsProxySendToServer** deviate from the [C706] specification of the Network Data Representation. This section documents how these two calls bypass NDR64 and how the data is represented on the wire. For more information about NDR64, see [MS-RPCE] section 2.2.5.

In the case of **TsProxySetupReceivePipe** and **TsProxySendToServer**, the Stub Data is not encoded using NDR64, instead it is sent over the wire as it is. Verification Trailer ([MS-RPCE] section 2.2.2.13) is also not passed with the Stub Data.

TsProxySetupReceivePipe and **TsProxySendToServer** modify the RPC Stub Data. Other elements, represented in the below table, are not modified.

Elements that are not modified:

Ethernet

IPv4

IPv6

TCP

HTTP

RPC

RPC Stub Data

RPC

3.3.1 TsProxySendToServer Request

The wire representation of the Stub Data in case of a TsProxySendToServer request is defined as follows.

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
Context Handle																															
...																															
...																															
...																															
...																															
Total Bytes																															
Number of Buffers																															
Buffer1 Length																															

Buffer2 Length (optional)
Buffer3 Length (optional)
Buffer1 (variable)
...
Buffer2 (variable)
...
Buffer3 (variable)
...

Context Handle (20 bytes): This field MUST be set to the context handle returned by a call to the TsProxyCreateChannel call. This context handle MUST be aligned to the 4-byte boundary.

Total Bytes (4 bytes): This field MUST be set to sum total of sizes of all the buffers and 4 bytes for each buffer. This is represented in the network byte order.

Number of Buffers (4 bytes): This field MUST be set to the total number of buffers. This MUST not exceed 0x00000003. This is represented in the network byte order.

Buffer1 Length (4 bytes): This field MUST be set to the length of the first buffer. This is represented in the network byte order

Buffer2 Length (4 bytes): This field MUST be set to the length of the first buffer. This is represented in the network byte order. If the Number of Buffers is set to 0x00000002 or 0x00000003, then this field is sent.

Buffer3 Length (4 bytes): This field MUST be set to the length of the first buffer. This is represented in the network byte order. If the Number of Buffers is set to 0x00000003, then this field is sent.

Buffer1 (variable): This field MUST contain the data corresponding to first buffer.

Buffer2 (variable): This field MUST contain the data corresponding to second buffer.

Buffer3 (variable): This field MUST contain the data corresponding to the third buffer.

3.3.2 TsProxySendToServer Response

The following is the response sent to the client.

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
ReturnValue																															

ReturnValue (4 bytes): Must be set to the return value of the TsProxySendToServer call.

3.3.3 TsProxySetupReceivePipe Request

The wire representation of the Stub Data in case of TsProxySetupReceivePipe request is as follows.

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
Context Handle																															
...																															
...																															
...																															
...																															

Context Handle (20 bytes): Must be set to the context handle returned by a call to the **TsProxyCreateChannel** call. This context handle must be aligned to the 4-byte boundary.

3.3.4 TsProxySetupReceivePipe Response

There can be multiple responses to the TsProxySetupReceivePipe call. Except for the last response, specified in section 3.3.5, the following is the representation of the Stub Data for all other responses.

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
Data (variable)																															
...																															

Data (variable): Must be set to the data to be sent to the TSG client. The size of this data is in the RPC header alloc_hint field specified in [C706].

3.3.5 TsProxySetupReceivePipe Final Response

The following represents the Stub data for the TsProxySetupReceivePipe call. For the final response PDU, the PFC_LAST_FRAG bit MUST be set in the **pfc_flags** field of the RPC response PDU.

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
ReturnValue																															

ReturnValue (4 bytes): Must be set to the return value of the call.

4 Protocol Examples

4.1 Normal Scenario

1. Initialization

   The TSG client obtains the name of a TSG server by using an out-of-band mechanism. The TSG client establishes a binding handle (a binding handle is specified in [C706] section 2.1) to the TSG server at the well-known endpoint of 443 and 3388.

2. The TSG server performs the authentication steps specified in section 2.1.

3. TSGU Protocol

   The TSG client then calls the **TsProxyCreateTunnel** method to create and obtain the tunnel context handle. As part of this call, the client sends current version capabilities to the server.

4. The TSG server receives the **TsProxyCreateTunnel** method. The TSG server authenticates the TSG client and uses policies to determine if the TSG client is allowed access to create a tunnel. The TSG server then creates a context handle to represent the tunnel and returns this to the TSG client. The server response includes the common capabilities of both the client and the server.

5. The TSG client makes the **TsProxyAuthorizeTunnel** method call using the tunnel context handle, optionally passing its health statement.

6. The TSG server receives **TsProxyAuthorizeTunnel** method call and verifies the tunnel context handle. The TSG server also performs RPC's verification and uses NAP policies to determine if the client is healthy. Assuming the TSG client is healthy, the TSG server returns success.

7. If both the client and the server are capable of handling administrative messages, the client can request for administrative message by making the **TsProxyMakeTunnelCall** method. This call is queued up on the server and is completed only when the said messages are available.

8. The TSG client makes the **TsProxyCreateChannel** method call using the tunnel context handle. The TSG client passes the target server information to the TSG server and obtains the channel context handle from the TSG server.

9. The TSG server receives the **TsProxyCreateChannel** method and determines, based on the NAP policy, if the TSG client is allowed to connect to the target server. If the connection is allowed, the TSG server creates a context handle to represent the channel and returns this to the TSG client.

10. The TSG client makes the **TsProxySetupReceivePipe** method call.

11. The TSG server receives the **TsProxySetupReceivePipe** method and creates an RPC out pipe. The TSG server can now send data on the pipe.

12. The TSG client and TSG server start sending and receiving data from this point.

13. The TSG client makes the **TsProxyCloseChannel** method call to close the channel.

14. The TSG server receives the **TsProxyCloseChannel** method and correctly closes the channel.

15. The TSG client then makes the **TsProxyCloseTunnel** method call to end the connection.

16. The TSG server receives the **TsProxyCloseTunnel** method and destroys the client connection.

For example, the client calls the **TsProxyCreateTunnel** method on a server named "fourthcoffee.example.com".

Example for the **TsProxyCreateTunnel** method:

```
HRESULT = {to be filled in by server}
TsProxyCreateTunnel(
    [in, ref] PTSG_PACKET tsgPacket;
    [out, ref] PTSG_PACKET* tsgPacketResponse =
      {to be filled in by server};
    [out] PTUNNEL_CONTEXT_HANDLE_SERIALIZE* tunnelContext =
      {to be filled in by server,
       and saved as m_tunnelcontext by client};
    [out] unsigned long* tunnelid =
      {to be filled in by server and saved as m_tunnelid by client};
    );
```

Where **TSG_PACKET** is set as follows.

```
typedef struct _TSG_PACKET
{
   unsigned long packetId = TSG_PACKET_TYPE_VERSIONCAPS;
   TSG_PACKET_TYPE_UNION tsgPacket {= packetVersionCaps};
} TSG_PACKET;
```

Where **TSG_PACKET_VERSIONCAPS** is filled in as follows.

```
typedef struct _TSG_PACKET_VERSIONCAPS
{
   TSG_PACKET_HEADER tsgHeader
      {
         ComponentId = 0x5452;
         PacketId = {unused};
      }
   PTSG_PACKET_CAPABILITIES tsgCaps
      {
         capabilityType = 1;
         tsgPacket.tsgCapNap = {1};
      }
   unsigned long numCapabilities = 1;
   unsigned short majorVersion = 1;
   unsigned short minorVersion = 1;
   unsigned short quarantineCapabilities = 0;
} TSG_PACKET_VERSIONCAPS;
```

The TSG server receives this method and returns the following.

```
HRESULT = S_OK
TsProxyCreateTunnel(
    [in, ref] PTSG_PACKET tsgPacket = {unchanged};
    [out, ref] PTSG_PACKET* tsgPacketResponse = =
       {filled in as shown below};
    [out] PTUNNEL_CONTEXT_HANDLE_SERIALIZE* tunnelContext =
       pContextHandleObject;
    [out] unsigned long* tunnelId = 1;
    );
```

Where [TSG_PACKET_RESPONSE]() is set as follows.

```
typedef struct _TSG_PACKET
{
   unsigned long packetId = TSG_PACKET_TYPE_QUARENC_RESPONSE;
   TSG_PACKET_TYPE_UNION tsgPacket {= packetQuarEncResponse};
} TSG_PACKET;
```

Where the [TSG_PACKET_QUARENC_RESPONSE]() is set as follows.

```
typedef struct _TSG_PACKET_QUARENC_RESPONSE
{
   unsigned long flags = 0;
   unsigned long certChainLen = {number of characters in certChainData};
   wchar_t* certChainData = {certificate chain data};
   GUID nonce = CreateGuid();
   PTSG_PACKET_VERSIONCAPS versionCaps
      {
         TSG_PACKET_HEADER tsgHeader
         {
            ComponentId = 0x5452;
            PacketId = TSG_PACKET_TYPE_VERSIONCAPS;
         }
         PTSG_PACKET_CAPABILITIES tsgCaps
         {
            capabilityType = 1;
            tsgPacket.tsgCapNap = {1};
         }
         unsigned long numCapabilities = 1;
         unsigned short majorVersion = 1;
         unsigned short minorVersion = 1;
         unsigned short quarantineCapabilities = 0;
      }
} TSG_PACKET_QUARENC_RESPONSE;
```

Example for **TsProxyAuthorizeTunnel** method.

```
HRESULT = {to be filled in by server}
TsProxyAuthorizeTunnel(
    [in] PTUNNEL_CONTEXT_HANDLE_NOSERIALIZE tunnelContext =
       m_tunnelcontext;
    [in, ref] PTSG_PACKET tsgPacket;
    [out, ref] PTSG_PACKET* tsgPacketResponse =
       { to be filled in by server};
    );
```

Where **TSG_PACKET** is set as follows.

```
typedef struct _TSG_PACKET
{
   unsigned long packetId = TSG_PACKET_TYPE_QUARREQUEST;
   TSG_PACKET_TYPE_UNION tsgPacket
      {=PTSG_PACKET_QUARREQUEST packetQuarRequest};
} TSG_PACKET;
```

Where the **TSG_PACKET_QUARREQUEST** is set as follows.

```
typedef struct _TSG_PACKET_QUARREQUEST
{
   unsigned long flags = 0;
   wchar_t* machineName = "mymachine";
   unsigned long nameLength = 10;
   byte *data = {statement of health prefixed with Nonce, which is received in response to
TsProxyCreateTunnel};
   unsigned long dataLen = {Number of bytes in the data field};
} TSG_PACKET_QUARREQUEST;
```

The TSG server receives this method and returns the following.

```
HRESULT = S_OK
TsProxyAuthorizeTunnel(
    [in] PTUNNEL_CONTEXT_HANDLE_NOSERIALIZE tunnelContext = unchanged;
    [in, ref] PTSG_PACKET tsgPacket = unchanged;
    [out, ref] PTSG_PACKET* tsgPacketResponse= filled in as below;
    );
```

Where the **TSG_PACKET_RESPONSE** is set as follows.

```
typedef struct _TSG_PACKET
{
   unsigned long packetId = TSG_PACKET_TYPE_RESPONSE;
   TSG_PACKET_TYPE_UNION tsgPacket
      {=PTSG_PACKET_RESPONSE packetResponse};
} TSG_PACKET;
```

Where the **packetResponse** is set as follows.

```
typedef struct _TSG_PACKET_RESPONSE
{
   unsigned long flags = TSG_PACKET_TYPE_QUARREQUEST;
   unsigned long reserved = 0;
   byte *responseData = NULL;
   unsigned long responseDataLen = 0;
   TSG_REDIRECTION_FLAGS redirectionFlags = {0,0,0,0,0,0,0,0};
} TSG_PACKET_RESPONSE;
```

Example for the **TsProxyMakeTunnelCall** method.

```
HRESULT = {to be filled in by server}
TsProxyMakeTunnelCall(
   [in] PTUNNEL_CONTEXT_HANDLE_NOSERIALIZE tunnelContext = m_tunnelcontext;
   [in] unsigned long procId,
   [in, ref] PTSG_PACKET tsgPacket,
   [out, ref] PTSG_PACKET* tsgPacketResponse = { to be filled in by server }
);
```

Where the procId and tsgPacket are set as follows.

```
procId = TSG_TUNNEL_CALL_ASYNC_MSG_REQUEST = 0x1
typedef struct _TSG_PACKET
{
   unsigned long packetId = TSG_PACKET_TYPE_MSGREQUEST_PACKET;
   TSG_PACKET_TYPE_UNION tsgPacket
      {=PTSG_PACKET_MSG_REQUEST packetMsgRequest};
} TSG_PACKET;
```

Where the TSG_PACKET_MSG_REQUEST is set as follows.

```
typedef struct _TSG_PACKET_MSG_REQUEST
    {
        unsigned long maxMessagesPerBatch = 1;
    } TSG_PACKET_MSG_REQUEST;
```

The TSG server receives this method and returns:

```
HRESULT = S_OK
TsProxyMakeTunnelCall(
   [in] PTUNNEL_CONTEXT_HANDLE_NOSERIALIZE tunnelContext = unchanged;
   [in] unsigned long procId = unchanged,
   [in, ref] PTSG_PACKET tsgPacket = unchanged,
   [out, ref] PTSG_PACKET* tsgPacketResponse = { filled in as below }
);
```

Where the TSG_PACKET_RESPONSE is set as follows.

```
typedef struct _TSG_PACKET
{
   unsigned long packetId = TSG_PACKET_TYPE_MESSAGE_PACKET;
   TSG_PACKET_TYPE_UNION tsgPacket
      {=PTSG_PACKET_MSG_RESPONSE packetMsgResponse};
} TSG_PACKET;
```

Where the packetMsgResponse is set as follows.

```
typedef struct _TSG_PACKET_MSG_RESPONSE
    {
        unsigned long msgID = 1;
        unsigned long msgType = TSG_ASYNC_MESSAGE_SERVICE_MESSAGE = 2;
        long isMsgPresent = 1;
        [switch_is(msgType)] TSG_PACKET_TYPE_MESSAGE_UNION messagePacket;
    } TSG_PACKET_MSG_RESPONSE;
```

Where the messagePacket is set as follows.

```
typedef [switch_type(unsigned long)] union
    {
        [case (TSG_ASYNC_MESSAGE_CONSENT_MESSAGE)]
            PTSG_PACKET_STRING_MESSAGE consentMessage;
        [case (TSG_ASYNC_MESSAGE_SERVICE_MESSAGE)]
            PTSG_PACKET_STRING_MESSAGE serviceMessage;
        [case (TSG_ASYNC_MESSAGE_REAUTH)]
            PTSG_PACKET_REAUTH_MESSAGE reauthMessage;
    } TSG_PACKET_TYPE_MESSAGE_UNION;
```

Where the servicemessage is set as follows.

```
typedef struct _TSG_PACKET_STRING_MESSAGE
    {
        long isDisplayMandatory = 1;
        long isConsentMandatory = 1;
        [range(0, 65536)] unsigned long msgBytes = 4;
        [size_is(msgBytes)] wchar_t* msgBuffer = "Test";
    } TSG_PACKET_STRING_MESSAGE;
```

Example for the **TsProxyCreateChannel** method.

```
HRESULT = {to be filled in by server}
TsProxyCreateChannel(
   [in] PTUNNEL_CONTEXT_HANDLE_NOSERIALIZE tunnelContext =
     m_tunnelcontext;
   [in, ref] PTSENDPOINTINFO tsEndPointInfo;
   [out] PCHANNEL_CONTEXT_HANDLE_SERIALIZE* channelContext =
     { to be filled in by server};
   [out] unsigned long* channelId = { to be filled in by server};
);
```

Where the **tsEndPointInfo** is set as follows.

```
typedef struct _tsendpointinfo
{
   RESOURCENAME *resourceNames = "myTsMachine";
   unsigned long numResourceNames = 1;
   RESOURCENAME *alternateResourceNames = NULL;
   unsigned short numAlternateResourceNames = 0;
   unsigned long Port = 222101507;
}TSENDPOINTINFO;
```

The TSG server receives this method and returns:

```
HRESULT = S_OK
TsProxyCreateChannel(
   [in] PTUNNEL_CONTEXT_HANDLE_NOSERIALIZE tunnelContext = unchanged;
   [in, ref] PTSENDPOINTINFO tsEndPointInfo = unchanged;
   [out] PCHANNEL_CONTEXT_HANDLE_SERIALIZE* channelContext =
      pServerChannelContextHandle;
   [out] unsigned long* channelId = 1;
);
```

Example for the **TsProxySendToServer** method.

```
DWORD = {to be filled in by server}
TsProxySendToServer(
   [in] TSG_SEND_MESSAGE_tsgSendMessage;
 );
```

Where the Generic Send Data Message Packet is as follows.

```
 m_channelContextHandle = {00 00 00 00 36 41 18
     41 dd 2d 84 43 83 63 82 cc b6 ea f3 f9 };

typedef struct _TSG_SEND_MESSAGE
{
   PTUNNEL_CONTEXT_HANDLE_NOSERIALIZE m_channelContextHandle; //as above
   DWORD totalDataLength = 0x00000008; //buffer1Length+sizeof(buffer1Length)
   DWORD numBuffers = 0x00000001; //number of buffers that follow is 1
   DWORD buffer1Length=0x04; //length of data that follows is 4 bytes
   PBYTE buffer1 = {04,00,00,03}; //data of 4 bytes
} TSG_SEND_MESSAGE;
```

The TSG server receives this method, verifies **m_channelContextHandle**, and sends the **buffer1Length** of **buffer1** to the target server and returns the following.

```
DWORD = ERROR_SUCCESS
TsProxySendToServer (
   [in] TSG_SEND_MESSAGE_tsgSendMessage = unchanged;
);
```

Example for the **TsProxySetupReceivePipe** method.

```
DWORD = {to be filled in by server}
TsProxySetupReceivePipe (
    [in, max_is(32767)] byte pRpcMessage[]
 );
```

Where an example value of pRpcMessage is as follows.

```
{
00 00 00 00 EC EC 2E 7D DB E2 E3 4A AE 61 A3 51 DC 53 55 61
}
```

The TSG server receives this method, sets up the out pipe, streams all necessary data to the TSG client in RPC response PDUs without setting the PFC_LAST_FRAG bit in the **pfc_flags** field, and when the TSG client calls **TsProxyCloseChannel** or calls **TsProxyCloseTunnel** without calling **TsProxyCloseChannel**, it returns the following return code in an RPC response PDU with PFC_LAST_FRAG bit set in the **pfc_flags** field.

```
DWORD = ERROR_GRACEFUL_DISCONNECT
TsProxySetupReceivePipe (
    [in, max_is(32767)] byte pRpcMessage[] = unchanged
);
```

Example for the **TsProxyCloseChannel** method.

```
HRESULT = {to be filled in by server}
TsProxyCloseChannel (
   [in, out] PCHANNEL_CONTEXT_HANDLE_NOSERIALIZE* context =
      m_channelContext;
 );
```

The TSG server receives this method and returns:

```
HRESULT = S_OK
TsProxyCloseChannel(
    [in, out] PCHANNEL_CONTEXT_HANDLE_NOSERIALIZE* context = NULL;
);
```

Example for the **TsProxyCloseTunnel** method.

```
HRESULT = {to be filled in by server}
TsProxyCloseTunnel (
   [in, out] PTUNNEL_CONTEXT_HANDLE_SERIALIZE* context =
      m_tunnelContext;
 );
```

The TSG server receives this method and returns:

```
HRESULT = S_OK
TsProxyCloseTunnel(
    [in, out] PTUNNEL_CONTEXT_HANDLE_SERIALIZE* context = NULL;
);
```

4.2 Pluggable Authentication Scenario with Consent Message Returned

- The TSG client obtains the name of a TSG server by using an out-of-band mechanism. The TSG client also obtains the cookie required for authenticating the user on the server by an out-of-band mechanism. The TSG client establishes a binding handle (a binding handle is specified in [C706] section 2.1) to the TSG server at the well-known endpoint of 443 and 3388. The TSG client then calls the **TsProxyCreateTunnel** method to create and obtain the tunnel context handle. It may be noted that at this point in time, the connection is unauthenticated. The TSG server then authenticates the user using the cookie that is passed in. As part of this call, the client sends current version capabilities to the server.
- The rest of the call flow is identical to what is specified in section 4.1.

For example, the client calls the **TsProxyCreateTunnel** method on a server named "fourthcoffee.example.com". The cookie content "Test" is used for authenticating the user. The Consent Message "Accept" is returned.

Example for the **TsProxyCreateTunnel** method:

```
HRESULT = {to be filled in by server}
TsProxyCreateTunnel(
    [in, ref] PTSG_PACKET tsgPacket;
    [out, ref] PTSG_PACKET* tsgPacketResponse =
      {to be filled in by server};
    [out] PTUNNEL_CONTEXT_HANDLE_SERIALIZE* tunnelContext =
      {to be filled in by server,
       and saved as m_tunnelcontext by client};
    [out] unsigned long* tunnelid =
      {to be filled in by server and saved as m_tunnelid by client};
    );
```

Where TSG_PACKET is set as follows.

```
typedef struct _TSG_PACKET
{
   unsigned long packetId = TSG_PACKET_TYPE_AUTH;
   TSG_PACKET_TYPE_UNION tsgPacket {= packetAuth};
} TSG_PACKET;
```

Where TSG_PACKET_AUTH is filled in as follows.

```
typedef struct _TSG_PACKET_AUTH
    {
        TSG_PACKET_VERSIONCAPS tsgVersionCaps;
        [range(0, 65536)]unsigned long cookieLen = 4;
        [size_is(cookieLen)]byte* cookie = "Test";
    } TSG_PACKET_AUTH;
```

Where TSG_PACKET_VERSIONCAPS is filled in as follows.

```
typedef struct _TSG_PACKET_VERSIONCAPS
{
   TSG_PACKET_HEADER tsgHeader
      {
         ComponentId = 0x5452;
         PacketId = TSG_PACKET_TYPE_VERSIONCAPS;
      }
   PTSG_PACKET_CAPABILITIES tsgCaps
      {
         capabilityType = 1;
         tsgPacket.tsgCapNap = {1};
      }
   unsigned long numCapabilities = 1;
   unsigned short majorVersion = 1;
   unsigned short minorVersion = 1;
   unsigned short quarantineCapabilities = 0;
} TSG_PACKET_VERSIONCAPS;
```

The TSG server receives this method and returns the following.

```
HRESULT = S_OK
TsProxyCreateTunnel(
    [in, ref] PTSG_PACKET tsgPacket = {unchanged};
    [out, ref] PTSG_PACKET* tsgPacketResponse = =
       {filled in as shown below};
    [out] PTUNNEL_CONTEXT_HANDLE_SERIALIZE* tunnelContext =
       pContextHandleObject;
    [out] unsigned long* tunnelId = 1;
    );
```

Where TSG_PACKET_RESPONSE is set as follows.

```
typedef struct _TSG_PACKET
{
   unsigned long packetId = TSG_PACKET_TYPE_CAPS_RESPONSE;
   TSG_PACKET_TYPE_UNION tsgPacket {= packetCapsResponse};
} TSG_PACKET;
```

Where the TSG_PACKET_CAPS_RESPONSE is set as follows.

```
typedef struct _TSG_PACKET_CAPS_RESPONSE
    {
        TSG_PACKET_QUARENC_RESPONSE pktQuarEncResponse;
        TSG_PACKET_MSG_RESPONSE pktConsentMessage;
    } TSG_PACKET_CAPS_RESPONSE;
```

Where the TSG_PACKET_QUARENC_RESPONSE is set as follows.

```
typedef struct _TSG_PACKET_QUARENC_RESPONSE
{
   unsigned long flags = 0;
   unsigned long certChainLen = 0;
   wchar_t* certChainData = "";
   GUID nonce = CreateGuid();
   PTSG_PACKET_VERSIONCAPS versionCaps
      {
         TSG_PACKET_HEADER tsgHeader
         {
            ComponentId = 0x5452;
            PacketId = TSG_PACKET_TYPE_VERSIONCAPS;
         }
         PTSG_PACKET_CAPABILITIES tsgCaps
         {
            capabilityType = 1;
            tsgPacket.tsgCapNap = {1};
         }
         unsigned long numCapabilities = 1;
         unsigned short majorVersion = 1;
         unsigned short minorVersion = 1;
         unsigned short quarantineCapabilities = 0;
      }
} TSG_PACKET_QUARENC_RESPONSE;
```

Where the TSG_PACKET_MSG_RESPONSE is set as follows.

```
    typedef struct _TSG_PACKET_MSG_RESPONSE
    {
        unsigned long msgID = 1;
        unsigned long msgType = TSG_ASYNC_MESSAGE_CONSENT_MESSAGE = 1;
        long isMsgPresent = 1;
        [switch_is(msgType)] TSG_PACKET_TYPE_MESSAGE_UNION messagePacket;
    } TSG_PACKET_MSG_RESPONSE;
```

Where the msgPacket is set as follows.

```
typedef [switch_type(unsigned long)] union
    {
        [case (TSG_ASYNC_MESSAGE_CONSENT_MESSAGE)]
PTSG_PACKET_STRING_MESSAGE consentMessage;
        [case (TSG_ASYNC_MESSAGE_SERVICE_MESSAGE)]
PTSG_PACKET_STRING_MESSAGE serviceMessage;
        [case (TSG_ASYNC_MESSAGE_REAUTH)]
PTSG_PACKET_REAUTH_MESSAGE reauthMessage;
    } TSG_PACKET_TYPE_MESSAGE_UNION;
```

Where the consentMessage is set as follows.

```
typedef struct _TSG_PACKET_STRING_MESSAGE
    {
        long isDisplayMandatory = 1;
        long isConsentMandatory = 1;
        [range(0, 65536)] unsigned long msgBytes = 7;
        [size_is(msgBytes)] wchar_t* msgBuffer = "Accept";
    } TSG_PACKET_STRING_MESSAGE;
```

4.3 Reauthentication

- Reauthentication is possible only if both the client and the server have the capability to handle the same. This capability is found out during the capability exchange during tunnel creation. This capability is based on capability to support Service Messages. As noted in section 4.1, a message request is queued up on the server using the **TsProxyMakeTunnelCall** method. The following sequence of calls takes place when the server expects the client to reauthenticate.
- The server completes the pending call. In the message type, it specifies that reauthentication is required. It also passes in the specific tunnel context so that when the client actually reauthenticates, the server can find out which client is doing the same.
- The client follows the steps 1, 2, 3, 4, 6, and 7 as specified in section 4.1. Only the initial packet is different, because it contains the tunnel context information that was passed back by the server.

The TSG server completes the pending TsProxyMakeTunnel calls as follows.

```
HRESULT = S_OK
TsProxyMakeTunnelCall(
   [in] PTUNNEL_CONTEXT_HANDLE_NOSERIALIZE tunnelContext = unchanged;
   [in] unsigned long procId = unchanged,
   [in, ref] PTSG_PACKET tsgPacket = unchanged,
   [out, ref] PTSG_PACKET* tsgPacketResponse = { filled in as below }
);
```

Where the TSG_PACKET_RESPONSE is set as follows.

```
typedef struct _TSG_PACKET
{
   unsigned long packetId = TSG_PACKET_TYPE_MESSAGE_PACKET;
   TSG_PACKET_TYPE_UNION tsgPacket
      {=PTSG_PACKET_MSG_RESPONSE packetMsgResponse};
} TSG_PACKET;
```

Where the packetMsgResponse is set as follows.

```
typedef struct _TSG_PACKET_MSG_RESPONSE
    {
        unsigned long msgID = 1;
        unsigned long msgType = TSG_ASYNC_MESSAGE_REAUTH = 3;
        long isMsgPresent = 1;
        [switch_is(msgType)] TSG_PACKET_TYPE_MESSAGE_UNION messagePacket;
    } TSG_PACKET_MSG_RESPONSE;
```

Where the messagePacket is set as follows.

```
typedef [switch_type(unsigned long)] union
    {
        [case (TSG_ASYNC_MESSAGE_CONSENT_MESSAGE)]
PTSG_PACKET_STRING_MESSAGE consentMessage;
        [case (TSG_ASYNC_MESSAGE_SERVICE_MESSAGE)]
PTSG_PACKET_STRING_MESSAGE serviceMessage;
        [case (TSG_ASYNC_MESSAGE_REAUTH)]
PTSG_PACKET_REAUTH_MESSAGE reauthMessage;
    } TSG_PACKET_TYPE_MESSAGE_UNION;
```

Where the reauthPacket is set as follows.

```
typedef struct _TSG_PACKET_REAUTH_MESSAGE
    {
        __int64 tunnelContext = 0x00123456;
    } TSG_PACKET_REAUTH_MESSAGE, *PTSG_PACKET_REAUTH_MESSAGE;
```

The client responds with the following call.

```
HRESULT = {to be filled in by server}
TsProxyCreateTunnel(
    [in, ref] PTSG_PACKET tsgPacket;
    [out, ref] PTSG_PACKET* tsgPacketResponse =
      {to be filled in by server};
    [out] PTUNNEL_CONTEXT_HANDLE_SERIALIZE* tunnelContext =
      {to be filled in by server,
       and saved as m_tunnelcontext by client};
    [out] unsigned long* tunnelid =
      {to be filled in by server and saved as m_tunnelid by client};
    );
```

Where TSG_PACKET is set as follows.

```
typedef struct _TSG_PACKET
{
   unsigned long packetId = TSG_PACKET_TYPE_REAUTH;
   TSG_PACKET_TYPE_UNION tsgPacket {= packetReauth};
} TSG_PACKET;
```

Where packetReauth is set as follows.

```
typedef struct _TSG_PACKET_REAUTH
    {
        __int64 tunnelContext = 0x00123456;
        unsigned long packetId = 0x5250;
        [switch_is(packetId)] TSG_INITIAL_PACKET_TYPE_UNION tsgInitialPacket;
    } TSG_PACKET_REAUTH, *PTSG_PACKET_REAUTH;
```

Where tsgInitialPacket is set as follows.

```
typedef [switch_type(unsigned long)] union
    {
        [case (TSG_PACKET_TYPE_VERSIONCAPS)]
            PTSG_PACKET_VERSIONCAPS packetVersionCaps;
        [case (TSG_PACKET_TYPE_AUTH)]
            PTSG_PACKET_AUTH packetAuth;
    } TSG_INITIAL_PACKET_TYPE_UNION;
```

Where TSG_PACKET_VERSIONCAPS is filled in as follows.

packetVersionCaps has been specified in section 4.1 and packetAuth in section 4.2.

5 Security

The following sections specify security considerations for implementers of the Terminal Services Gateway Server Protocol and an index of security parameters.

5.1 Security Considerations for Implementers

Authenticated RPC should be used by this protocol, as specified in [C706] section 13.

The TSG server should audit all tunnel and channel connections to the target server. The TSG server should have policies that determine which TSG clients are allowed to connect and which authentication service they use. A two-factor authentication should be required due to the nature of the deployment for this protocol, which is typically at the neutral zone. The TSG server should have policies that determine which TSG clients are allowed to connect to which target servers. Deployments should also consider having a front-end security mechanism such as an outside firewall before allowing connections to the TSG server.

During the **TsProxyCreateTunnel**, the TSG server sends a nonce represented by a GUID to uniquely identify the connection to prevent statement of health (SoH) replay attacks. The TSG client MUST send this GUID if it sends the statement of health, as specified in section 2.2.3.2.1.4.

5.2 Index of Security Parameters

Security parameter	Section
Authentication service settings	2.1

6 Appendix A: Full IDL

For ease of implementation, the full **IDL** is provided below, where "ms-dtyp.idl" is the IDL as specified in [MS-DTYP] Appendix A.

```
import "ms-dtyp.idl";

[
    uuid(44e265dd-7daf-42cd-8560-3cdb6e7a2729),
    version(1.3),
    pointer_default(unique)
]

interface TsProxyRpcInterface
{
    typedef [context_handle] void*
        PTUNNEL_CONTEXT_HANDLE_NOSERIALIZE;

    typedef [context_handle] void*
        PCHANNEL_CONTEXT_HANDLE_NOSERIALIZE;

    typedef [context_handle]
    PTUNNEL_CONTEXT_HANDLE_NOSERIALIZE
    PTUNNEL_CONTEXT_HANDLE_SERIALIZE;

    typedef [context_handle]
    PCHANNEL_CONTEXT_HANDLE_NOSERIALIZE
    PCHANNEL_CONTEXT_HANDLE_SERIALIZE;

    typedef [string] wchar_t* RESOURCENAME;

#define MAX_RESOURCE_NAMES 50

    typedef struct _tsendpointinfo {
        [size_is(numResourceNames)] RESOURCENAME* resourceName;
        [range(0, MAX_RESOURCE_NAMES)]
            unsigned long numResourceNames;
        [unique, size_is(numAlternateResourceNames)]
            RESOURCENAME* alternateResourceNames;
        [range(0, 3)]
            unsigned short numAlternateResourceNames;
        unsigned long Port;
    } TSENDPOINTINFO,
     *PTSENDPOINTINFO;

#define TSG_PACKET_TYPE_HEADER              0x00004844
#define TSG_PACKET_TYPE_VERSIONCAPS         0x00005643
#define TSG_PACKET_TYPE_QUARCONFIGREQUEST   0x00005143
#define TSG_PACKET_TYPE_QUARREQUEST         0x00005152
#define TSG_PACKET_TYPE_RESPONSE            0x00005052
#define TSG_PACKET_TYPE_QUARENC_RESPONSE    0x00004552
#define TSG_CAPABILITY_TYPE_NAP             0x00000001
#define TSG_PACKET_TYPE_CAPS_RESPONSE       0x00004350
#define TSG_PACKET_TYPE_MSGREQUEST_PACKET   0x00004752
#define TSG_PACKET_TYPE_MESSAGE_PACKET      0x00004750
#define TSG_PACKET_TYPE_AUTH                0x00004054
#define TSG_PACKET_TYPE_REAUTH              0x00005250
```

```
#define TSG_ASYNC_MESSAGE_CONSENT_MESSAGE   0x00000001
#define TSG_ASYNC_MESSAGE_SERVICE_MESSAGE   0x00000002
#define TSG_ASYNC_MESSAGE_REAUTH            0x00000003
#define TSG_TUNNEL_CALL_ASYNC_MSG_REQUEST   0x00000001
#define TSG_TUNNEL_CANCEL_ASYNC_MSG_REQUEST 0x00000002



    typedef struct _TSG_PACKET_HEADER {
        unsigned short ComponentId;
        unsigned short PacketId;
    } TSG_PACKET_HEADER,
     *PTSG_PACKET_HEADER;

    typedef struct _TSG_CAPABILITY_NAP{
        unsigned long capabilities;
    } TSG_CAPABILITY_NAP,
     *PTSG_CAPABILITY_NAP;

    typedef [switch_type(unsigned long)] union {
        [case (TSG_CAPABILITY_TYPE_NAP)]
        TSG_CAPABILITY_NAP tsgCapNap;
    } TSG_CAPABILITIES_UNION,
     *PTSG_CAPABILITIES_UNION;

    typedef struct _TSG_PACKET_CAPABILITIES {
        unsigned long capabilityType;
        [switch_is(capabilityType)]
        TSG_CAPABILITIES_UNION tsgPacket;
    } TSG_PACKET_CAPABILITIES,
     *PTSG_PACKET_CAPABILITIES;

    typedef struct _TSG_PACKET_VERSIONCAPS {
        TSG_PACKET_HEADER tsgHeader;
        [size_is(numCapabilities)]
            PTSG_PACKET_CAPABILITIES tsgCaps;
        [range(0, 32)] unsigned long numCapabilities;
        unsigned short majorVersion;
        unsigned short minorVersion;
        unsigned short quarantineCapabilities;
    } TSG_PACKET_VERSIONCAPS,
     *PTSG_PACKET_VERSIONCAPS;

    typedef struct _TSG_PACKET_QUARCONFIGREQUEST {
        unsigned long flags;
    } TSG_PACKET_QUARCONFIGREQUEST,
     *PTSG_PACKET_QUARCONFIGREQUEST;

    typedef struct _TSG_PACKET_QUARREQUEST {
        unsigned long flags;
        [string, size_is(nameLength)] wchar_t* machineName;
        [range(0, 512 + 1)] unsigned long nameLength;
        [unique, size_is(dataLen)]  byte* data;
        [range(0, 8000)] unsigned long dataLen;
    } TSG_PACKET_QUARREQUEST,
     *PTSG_PACKET_QUARREQUEST;

    typedef struct _TSG_REDIRECTION_FLAGS {
        BOOL enableAllRedirections;
```

```
            BOOL disableAllRedirections;
            BOOL driveRedirectionDisabled;
            BOOL printerRedirectionDisabled;
            BOOL portRedirectionDisabled;
            BOOL reserved;
            BOOL clipboardRedirectionDisabled;
            BOOL pnpRedirectionDisabled;
        } TSG_REDIRECTION_FLAGS,
         *PTSG_REDIRECTION_FLAGS;

        typedef struct _TSG_PACKET_RESPONSE {
            unsigned long flags;
            unsigned long reserved;
            [size_is(responseDataLen)] byte* responseData;
            [range(0, 24000)] unsigned long responseDataLen;
            TSG_REDIRECTION_FLAGS redirectionFlags;
        } TSG_PACKET_RESPONSE,
         *PTSG_PACKET_RESPONSE;

        typedef struct _TSG_PACKET_QUARENC_RESPONSE {
            unsigned long flags;
            [range(0, 24000)] unsigned long certChainLen;
            [string, size_is(certChainLen)] wchar_t* certChainData;
            GUID nonce;
            PTSG_PACKET_VERSIONCAPS versionCaps;
        } TSG_PACKET_QUARENC_RESPONSE,
         *PTSG_PACKET_QUARENC_RESPONSE;

    typedef struct _TSG_PACKET_MSG_REQUEST {
    unsigned long maxMessagesPerBatch;
    } TSG_PACKET_MSG_REQUEST, *PTSG_PACKET_MSG_REQUEST;


    typedef struct _TSG_PACKET_STRING_MESSAGE {
      long isDisplayMandatory;
      long isConsentMandatory;
      [range(0,65536)] unsigned long msgBytes;
      [size_is(msgBytes)] wchar_t* msgBuffer;
    } TSG_PACKET_STRING_MESSAGE,
     *PTSG_PACKET_STRING_MESSAGE;

    typedef struct _TSG_PACKET_REAUTH_MESSAGE {
    unsigned __int64 tunnelContext;
    } TSG_PACKET_REAUTH_MESSAGE, *PTSG_PACKET_REAUTH_MESSAGE;

    typedef
    [switch_type(unsigned long)]
    union {
      [case(TSG_ASYNC_MESSAGE_CONSENT_MESSAGE)]
    PTSG_PACKET_STRING_MESSAGE consentMessage;
      [case(TSG_ASYNC_MESSAGE_SERVICE_MESSAGE)]
    PTSG_PACKET_STRING_MESSAGE serviceMessage;
      [case(TSG_ASYNC_MESSAGE_REAUTH)]
    PTSG_PACKET_REAUTH_MESSAGE reauthMessage;
    } TSG_PACKET_TYPE_MESSAGE_UNION,
     *PTSG_PACKET_TYPE_MESSAGE_UNION ;

    typedef struct _TSG_PACKET_MSG_RESPONSE {
    unsigned long msgID;
```

```
unsigned long msgType;
long isMsgPresent;
[switch_is(msgType)] TSG_PACKET_TYPE_MESSAGE_UNION messagePacket;
} TSG_PACKET_MSG_RESPONSE,
*PTSG_PACKET_MSG_RESPONSE;

typedef struct _TSG_PACKET_CAPS_RESPONSE {
TSG_PACKET_QUARENC_RESPONSE pktQuarEncResponse;
TSG_PACKET_MSG_RESPONSE pktConsentMessage;
} TSG_PACKET_CAPS_RESPONSE, *PTSG_PACKET_CAPS_RESPONSE;

typedef struct _TSG_PACKET_AUTH {
  TSG_PACKET_VERSIONCAPS tsgVersionCaps;
  [range(0, 65536)] unsigned long cookieLen;
  [size_is(cookieLen)] byte* cookie;
} TSG_PACKET_AUTH, *PTSG_PACKET_AUTH;

typedef
[switch_type(unsigned long)]
union {
  [case(TSG_PACKET_TYPE_VERSIONCAPS)]
PTSG_PACKET_VERSIONCAPS packetVersionCaps;
  [case(TSG_PACKET_TYPE_AUTH)]
PTSG_PACKET_AUTH packetAuth;
} TSG_INITIAL_PACKET_TYPE_UNION,
 *PTSG_INITIAL_PACKET_TYPE_UNION;

typedef struct _TSG_PACKET_REAUTH {
  unsigned __int64 tunnelContext;
  unsigned long packetId;
  [switch_is(packetId)] TSG_INITIAL_PACKET_TYPE_UNION tsgInitialPacket;
} TSG_PACKET_REAUTH,
 *PTSG_PACKET_REAUTH;

typedef [switch_type(unsigned long)] union {
        [case (TSG_PACKET_TYPE_HEADER)]
            PTSG_PACKET_HEADER packetHeader;
        [case (TSG_PACKET_TYPE_VERSIONCAPS)]
            PTSG_PACKET_VERSIONCAPS packetVersionCaps;
        [case (TSG_PACKET_TYPE_QUARCONFIGREQUEST)]
            PTSG_PACKET_QUARCONFIGREQUEST packetQuarConfigRequest;
        [case (TSG_PACKET_TYPE_QUARREQUEST)]
            PTSG_PACKET_QUARREQUEST packetQuarRequest;
        [case (TSG_PACKET_TYPE_RESPONSE)]
            PTSG_PACKET_RESPONSE packetResponse;
        [case (TSG_PACKET_TYPE_QUARENC_RESPONSE)]
            PTSG_PACKET_QUARENC_RESPONSE packetQuarEncResponse;
        [case (TSG_PACKET_TYPE_CAPS_RESPONSE)]
            PTSG_PACKET_CAPS_RESPONSE packetCapsResponse;
        [case (TSG_PACKET_TYPE_MSGREQUEST_PACKET)]
            PTSG_PACKET_MSG_REQUEST packetMsgRequest;
        [case (TSG_PACKET_TYPE_MESSAGE_PACKET)]
            PTSG_PACKET_MSG_RESPONSE packetMsgResponse;
        [case (TSG_PACKET_TYPE_AUTH)]
            PTSG_PACKET_AUTH packetAuth;
        [case (TSG_PACKET_TYPE_REAUTH)]
            PTSG_PACKET_REAUTH packetReauth;
    } TSG_PACKET_TYPE_UNION,
     *PTSG_PACKET_TYPE_UNION;
    typedef struct _TSG_PACKET {
        unsigned long packetId;
        [switch_is(packetId)] TSG_PACKET_TYPE_UNION tsgPacket;
    } TSG_PACKET,
     *PTSG_PACKET;


    void Opnum0NotUsedOnWire(void);

    HRESULT
    TsProxyCreateTunnel(
        [in, ref] PTSG_PACKET tsgPacket,
        [out, ref] PTSG_PACKET* tsgPacketResponse,
        [out] PTUNNEL_CONTEXT_HANDLE_SERIALIZE* tunnelContext,
        [out] unsigned long* tunnelId
    );

    HRESULT
    TsProxyAuthorizeTunnel(
        [in] PTUNNEL_CONTEXT_HANDLE_NOSERIALIZE tunnelContext,
        [in, ref] PTSG_PACKET tsgPacket,
        [out, ref] PTSG_PACKET* tsgPacketResponse
    );

   HRESULT
   TsProxyMakeTunnelCall(
      [in] PTUNNEL_CONTEXT_HANDLE_NOSERIALIZE tunnelContext,
      [in] unsigned long procId,
      [in, ref] PTSG_PACKET tsgPacket,
      [out, ref] PTSG_PACKET* tsgPacketResponse
   );

   HRESULT
   TsProxyCreateChannel(
       [in] PTUNNEL_CONTEXT_HANDLE_NOSERIALIZE tunnelContext,
       [in, ref] PTSENDPOINTINFO tsEndPointInfo ,
       [out] PCHANNEL_CONTEXT_HANDLE_SERIALIZE* channelContext,
       [out] unsigned long* channelId
   );

   void Opnum5NotUsedOnWire(void);

   HRESULT
   TsProxyCloseChannel(
       [in, out] PCHANNEL_CONTEXT_HANDLE_NOSERIALIZE* context
   );

   HRESULT
   TsProxyCloseTunnel(
       [in, out] PTUNNEL_CONTEXT_HANDLE_SERIALIZE* context
   );

//see section 2.2.3.3 for decoding instructions
   DWORD
   TsProxySetupReceivePipe(
       [in, max_is(32767)] byte pRpcMessage[]
   );
```

```
//see section 2.2.3.4 for decoding instructions
   DWORD
   TsProxySendToServer(
       [in, max_is(32767)] byte pRpcMessage[]
   );

};
```

7 Appendix B: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Windows® XP operating system Service Pack 2 (SP2)
- Windows Server® 2003 operating system with Service Pack 1 (SP1)
- Windows Vista® operating system
- Windows Server® 2008 operating system
- Windows® 7 operating system
- Windows Server® 2008 R2 operating system

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

<1> Section 1.3: The Microsoft RDP client uses the Terminal Services Gateway Server Protocol as a transport mechanism to establish a connection to a target server behind a firewall. The connection frequently originates from a client located on the Internet. Terminal Services Gateway Server Protocol may also be used to connect to an isolated target server from clients located on a different private network. A Terminal Services Gateway Server Protocol server serves as the termination point for the tunnel and will relay RDP client data to and from the target server by using the channel.

<2> Section 2.2.2.19: Only Windows 7 and Windows Server 2008 R2 support Consent Message, Service Message, Idle Timeout, and Reauthentication.

<3> Section 2.2.2.24: All TSG–supported versions of Windows use the identity of the caller to perform method-specific access checks. The TSG server allows only authenticated users to call any method. Windows Server 2008 imposes a minimum impersonation level of RPC_C_IMPL_LEVEL_IDENTIFY ([MS-RPCE] section 2.2.1.1.9) on all method calls. If the TSG server is operating in a load-balanced environment, Windows Server 2008 registers for the hostname, not **IPv4**/**IPv6** addresses. Windows Server 2008 registers for RPC_C_AUTHN_GSS_SCHANNEL Authentication Service (AS) using the same certificate that is set for HTTPS communications on the machine.

<4> Section 2.2.2.24: Windows Server 2008 does not attempt to connect to the target server during the TsProxyCreateChannel call. The actual connection to the target server happens during the call to **TsProxySetupReceivePipe**.

<5> Section 2.2.2.24: The Windows Server 2008 R2 Standard Edition implementation limits the number of connections to 250.

The Windows Server 2008 R2 Foundation Edition implementation limits the number of connections to 50.

All other Windows implementations allow an unlimited number of connections.

<6> Section 2.2.2.24: This error is returned only by the Windows Server 2008 TSG server, because only this version attempts connecting to the target server in the **TsProxySetupReceivePipe** call.

<7> Section 2.2.2.24: Windows 7 and Windows Server 2008 R2 are capable of exchanging policies with the TSG server. Windows XP SP2, Windows Server 2003 with SP1, Windows Vista, and Windows Server 2008 are not capable of exchanging policies with the TSG server.

<8> Section 2.2.3.1: Windows Server 2003 with SP1, Windows XP SP2, and Windows Vista send a list of IP addresses in the **resourceName** field and NetBIOS or FQDN names in **alternateResourceNames** when it is redirected by the TS session directory.

<9> Section 2.2.3.2.1.2: Windows XP SP2, Windows Vista, Windows Server 2003 with SP1, Windows Server 2008, and Windows Server 2008 R2 send quarantineCapabilities type 1—indicating that each understands network access protection capability. Based on quarantine policies set on Windows Server 2008, it will require quarantine information be sent from client to server.

<10> Section 2.2.3.2.1.2.1: Windows XP SP2, Windows Vista, Windows Server 2003 with SP1, and Windows Server 2008 send the capability type 0x00000001 indicating that each understands NAP capability. Based on quarantine policies set on Windows Server 2008, it will require quarantine information to be sent from client to server.

<11> Section 2.2.3.2.1.3: The TSG_PACKET_QUARCONFIGREQUEST structure is not used by any version of Windows. If this structure is used, an error code of HRESULT_CODE(E_PROXY_NOTSUPPORTED) is returned.

<12> Section 2.2.3.2.1.4: If Windows Server 2008 requires that quarantine information be sent, the client's health is queried using quarantine agent and is sent to the Windows Server 2008 in an encrypted manner. If this data is not present and quarantine is required by Windows Server 2008, the server rejects the **TsProxyAuthorizeTunnel** call with an E_PROXY_QUARANTINE_ACCESSDENIED (0x800759ED) response.

<13> Section 2.2.3.2.1.4: Windows Server 2008 uses machineName value to determine the machine domain membership based on the network access policies set by the administrator on the server.

<14> Section 2.2.3.2.1.4: Windows XP SP2, Windows Server 2003 with SP1, and Windows Vista obtain the statement of health from the NAP agent and encrypt it using the certificate sent by the server during the **TsProxyCreateTunnel** method. Windows Server 2008 decrypts the statement of health from the client using the private key corresponding to the same certificate it sent to the client during the tunnel creation. If the packet contains health data, Windows Server 2008 performs all access checks, including quarantine, and network policies in this call to allow operations on the tunnel.

<15> Section 2.2.3.2.1.5: In Windows Server 2008, **responseData** is ignored and **responseDataLen** is set to zero.

Windows Server 2008 R2 may send the statement of health response (SoHR) and idle timeout values, depending on its policies. The statement of health response is signed and encoded using the TSG server's private key. The TSG client sends the statement of health response to the NAP agent, which verifies and decodes the data using the server public key that was passed during a call to **TsProxyCreateTunnel**. If the TSG server can support idle timeout as specified in section 2.2.3.2.1.2.1.2, then the idle timeout is prepended to the statement of health response.

Idle timeout is configured on the TSG server and is enforced on the TSG client. Only Windows Server 2008 R2 TSG server supports idle timeout.

<16> Section 2.2.3.2.1.5: Windows Server 2008 sends the **redirectionFlags** value based on network policies configured for Windows terminal server. Regarding the details of redirectionFlag values please refer to section 2.2.1.27 of [MS-RNAP].

<17> Section 2.2.3.2.1.6: Windows Server 2008 sends the base64-encoded version of the certificate chain if quarantine is required. This certificate is the same as that registered for the RPC_C_AUTHN_GSS_SCHANNEL authentication service.

<18> Section 2.2.3.2.1.9: Windows implementation of TSG server always sets this field to 1 and Windows implementation of TSG client never uses this field.

<19> Section 3.1.1: On machines running Windows, this is the machine name that is returned by the **gethostname** function.

<20> Section 3.1.1: Windows Server 2003 with SP1, Windows Server 2008, and Windows Server 2008 R2 use Tunnel id to map to a Tunnel context handle, Channel id capabilities information, and user information.

<21> Section 3.1.1: Windows Server 2003 with SP1, Windows Server 2008, and Windows Server 2008 R2 use the Channel id for an auditing purpose at server side and to show the connection details to the administrator.

<22> Section 3.1.2.1: The Session Timeout Timer is not implemented in Windows XP SP2, Windows Server 2003 with SP1, Windows Vista, Windows Server 2008, and Windows 7.

<23> Section 3.1.2.2: The Re-authentication Timer is not implemented in Windows XP SP2, Windows Server 2003 with SP1, Windows Vista, Windows Server 2008, and Windows 7.

<24> Section 3.1.2.3: Windows Server 2008 implements this timer, but Windows Server 2008 R2 does not implement this timer. In Windows Server 2008, if a call to **TsProxySetupReceivePipe** is not made within 30 seconds of a call to **TsProxyCreateChannel**, the Windows Server 2008 TSG server will disconnect the connection. The disconnection will occur in order to implement **TsProxyCreateChannel**. Note that the protocol, however, does not mandate the timer.

<25> Section 3.1.2.3: The timer value is not mandated by the protocol. Different implementations may choose to use this timer if required. The timer value may be set to a value appropriate to the implementation.

<26> Section 3.1.3: Windows Server 2008 uses the identity of the caller to perform method-specific access checks. The TSG service allows only authenticated users to call any method. Windows Server 2008 imposes a minimum impersonation level of RPC_C_IMPL_LEVEL_IDENTIFY ([MS-RPCE] section 2.2.1.1.9) on all method calls. If TSG is operating in a load-balanced environment, Windows Server 2008 registers for the hostname, not the ipv4/ipv6 addresses. Windows Server 2008 registers for RPC_C_AUTHN_GSS_SCHANNEL authentication service using the same certificate that is set for HTTPS communications on the machine.

<27> Section 3.1.4: Windows Server 2008 implementation uses RPC protocol to retrieve the identity of the caller as specified in [MS-RPCE] section 3.2.3.4.2. The server uses the underlying Windows security subsystem to determine the permissions for the caller. If the caller does not have the required permissions to execute a specific method, the method call fails with ERROR_ACCESS_DENIED. This error code is returned to the caller in a rpc_fault packet.

<28> Section 3.1.4: This method is available only in Windows 7 and Windows Server 2008 R2.

<29> Section 3.1.4: Opnums that are not used apply to Windows XP SP2, Windows Vista, Windows Server 2003 with SP1, Windows Server 2008, Windows 7, and Windows Server 2008 R2.

Opnum 3 is used only by Windows 7 and Windows Server 2008 R2. Windows XP SP2 and Windows Server 2008 do not use opnum 3.

Opnum	Description
0	Reserved for local use.
5	Reserved for local use.

<30> Section 3.1.4.1.1: Pluggable authentication is available only in Windows 7 and Windows Server 2008 R2. Windows does not implement any authentication plugins, but ISVs can create their plugins and use them for authentication.

<31> Section 3.1.4.1.1: In Windows Server 2008, the results are undefined when the **tsgPacket** is set to anything other than the **TSG_PACKET_VERSIONCAPS** structure. However, in Windows Server 2008 R2, if the **tsgPacket** is set to anything other than the **TSG_PACKET_VERSIONCAPS** structure in case of RPC authentication or **TSG_PACKET_AUTH** structure in case of pluggable authentication, the error <E_PROXY_INTERNALERROR> is returned.

<32> Section 3.1.4.1.2: Windows implementation of TS Gateway protocol does user authorization based on user group membership, client computer group membership (optional), user authentication method (password or smartcard), and client computer health status (optional). These authorization conditions are specified using connection authorization policies (CAPs). When the CAPs set by the administrator require TSG client computer health status checks, the TSG server will require that TSG clients send health information and remediate themselves if health check is not met.

<33> Section 3.1.4.1.2: The Windows Server 2008 R2 Standard implementation limits the number of connections to 250.

The Windows Server 2008 R2 Foundation implementation limits the number of connections to 50.

All other Windows implementations allow an unlimited number of connections.

<34> Section 3.1.4.1.4: Windows Server 2008 rejects this call and all channel-related calls if the **TsProxyAuthorizeTunnel** method call does not succeed. Windows Server 2008 performs access checks to determine if a connection to the target server is allowed by policies in this call.

<35> Section 3.1.4.1.4: Windows Server 2008 does not attempt to connect to the target server during the TsProxyCreateChannel call. The actual connection to the target server happens during the call to TsProxySetupReceivePipe.

<36> Section 3.1.4.1.4: Windows Server 2008 returns HRESULT_CODE(E_PROXY_RAP_ACCESSDENIED), such as 0x000059DA, if resource authorization fails.

<37> Section 3.1.4.1.4: In Windows Server 2008, even if the **RESOURCENAME** strings in the **resourceName** member are not valid, ERROR_SUCCESS is returned. In Windows Server 2008 R2, if the **RESOURCENAME** is not valid, HRESULT_CODE(E_PROXY_TS_CONNECTFAILED) (0x000059DD) is returned.

<38> Section 3.1.4.2.1: Windows Server 2008, Windows Server 2003 with SP1, Windows XP SP2, and Windows Vista do not use the NDR for this call. Windows Server 2008 rejects this call if any discrepancies in the data are noted, such as the data lengths not matching those reported by the server stub.

<39> Section 3.1.4.2.2: To bypass NDR, the Windows implementation of Terminal Services Gateway Server Protocol hooks into the RPC layer directly and reads from the Buffer field of the _RPC_MESSAGE struct defined in [MSDN-RPCMESSAGE].

<40> Section 3.1.4.2.2: Windows Server 2008, Windows Server 2003 with SP1, Windows XP SP2, and Windows Vista do not use the NDR for this call. Windows Server 2003 with SP1, Windows XP SP2, Windows Vista, and Windows Server 2008 disable RPC buffering for this call. The Windows Server 2008 rejects this call if any discrepancies in the data are noted, such as the data lengths not matching those reported by the server stub. Windows Server 2008 makes a socket connection to the target server as part of this call.

<41> Section 3.1.4.2.2: Only Windows Server 2008 attempts to connect to the target server during the **TsProxySetupReceivePipe** call because it doesn't attempt to connect to the target server during **TsProxyCreateChannel** call.

<42> Section 3.1.4.2.2: This error is returned only by the Windows Server 2008 TSG server, because only this version attempts connecting to the target server in the **TsProxySetupReceivePipe** call.

<43> Section 3.2.1: On machines running Windows, this is the machine name that is returned by the **gethostname** function.

<44> Section 3.2.1: Note that the size of the buffer is 513 bytes, even though the contents are 16-bit Unicode characters. This reflects the actual Windows implementation.

<45> Section 3.2.1: On machines running Windows, the Client Machine name refers to the computer name only as returned by the **gethostname** function.

<46> Section 3.2.4: Windows uses the **INapEnforcementClientConnection::GetSoHRequest** method to obtain the SoH, which is retrieved in the out parameter as specified in [MSDN-NAPAPI].

8 Change Tracking

This section identifies changes that were made to the [MS-TSGU] protocol document between the January 2011 and February 2011 releases. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements or functionality.
- An extensive rewrite, addition, or deletion of major portions of content.
- The removal of a document from the documentation set.
- Changes made for template compliance.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **Editorial** means that the language and formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical or language changes were introduced. The technical content of the document is identical to the last released version, but minor editorial and formatting changes, as well as updates to the header and footer information, and to the revision summary, may have been made.

Major and minor changes can be described further using the following change types:

- New content added.
- Content updated.
- Content removed.
- New product behavior note added.
- Product behavior note updated.
- Product behavior note removed.
- New protocol syntax added.
- Protocol syntax updated.
- Protocol syntax removed.
- New content added due to protocol revision.
- Content updated due to protocol revision.
- Content removed due to protocol revision.
- New protocol syntax added due to protocol revision.

- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.
- New content added for template compliance.
- Content updated for template compliance.
- Content removed for template compliance.
- Obsolete document removed.

Editorial changes are always classified with the change type **Editorially updated.**

Some important terms used in the change type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.
- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

The changes made to this document are listed in the following table. For more information, please contact protocol@microsoft.com.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
1.1 Glossary	58585 Added the terms "statement of health (SoH)" and "statement of health response (SoHR)".	Y	Content updated.
1.1 Glossary	58583 Added "Triple Data Encryption Standard" to the list of terms defined in [MS-GLOS].	Y	Content updated.
1.1 Glossary	58586 Added "SHA-1 hash" to the list of terms defined in [MS-GLOS].	Y	Content updated.
1.2.1 Normative References	58582 Added reference to [MS-NAPSO].	Y	Content updated.
1.2.2 Informative References	58582 Added reference [MSDN-NAPAPI].	Y	Content updated.
2.2.3.2.1.5 TSG_PACKET_RESPONSE	58588 Changed "encrypted" to "signed and encoded" and "decrypts" to "verifies and decodes" in the product behavior note that describes that the servermay send the SoHR) and idle timeout values.	Y	Content updated.
3.1.4.1.2 TsProxyAuthorizeTunnel (Opnum 2)	58585 Added processing rules describing the decoding of the SoH request by the server and specified that the TSG server MUST verify the signature of	Y	Content updated.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
	the SoHR.		
3.1.4.1.2 TsProxyAuthorizeTunnel (Opnum 2)	58580 Added processing rules regarding the [MS-NAPSO] Proxy SoH Task.	Y	Content updated.
3.1.4.1.2 TsProxyAuthorizeTunnel (Opnum 2)	58586 Added processing rules regarding the SoHR encoding, and specified that the TSG server MUST sign the SoHR using an SHA1 hash.	Y	Content updated.
3.1.6 Other Local Events	58645 Added an abstract interface for sending an SoHR packet.	Y	Content updated.
3.2.4 Message Processing Events and Sequencing Rules	58584 Added processing rules regarding the transport details of the SoH request.	Y	Content updated.
3.2.4 Message Processing Events and Sequencing Rules	58583 Added processing rules regarding the SoH encryption, and specified that the TSG client MUST use the Triple Data Encryption Standard to encrypt the SoH.	Y	Content updated.
3.2.4 Message Processing Events and Sequencing Rules	58582 Added processing rules regarding getting the SoH and encrypting it with the TSG server certificate context. Added a reference to [MS-SOH] detailing the format of the SoH, and added a product behavior note.	Y	Content updated.
3.2.4 Message Processing Events and Sequencing Rules	58588 Added processing rules regarding the initialization of the Statement of health response variable. Changed the term "decrypt" to "decode" and added the explicit requirement that the TSG server verify the signature of the SoHR.	Y	Content updated.
3.2.4 Message Processing Events and Sequencing Rules	58581 Added processing rules regarding the decryption of the SoHR response.	Y	Content updated.
3.2.6 Other Local Events	58644 Specified when NAPSO informs the TSG client about changes in the TSG client computer's health and clarified that such notification does not result in an exchange of data between the TSG client and the TSG server.	Y	Content updated.

9 Index