[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.

Date	Revision History	Revision Class	Comments
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.

Table of 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, as specified in [MS-RDPBCGR].

1.1 Glossary

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

Authentication Level
Authentication Service (AS)
Certificate
Client
Endpoint
Globally Unique Identifier (GUID)
Handle
Interface Definition Language (IDL)
Network Data Representation (NDR)
Opnum
Remote Procedure Call (RPC)
Server
Universally Unique Identifier (UUID)
Well-Known Endpoint

The following terms are specific to this document:

Channel: A term indicating a successful connection between the TSG **client** and target **server**. A channel can only be established within the context of a **tunnel**. The channel is specific to TSG **client** and **tunnel** instance. The channel is stateful. Multiple channels can exist within a **tunnel**.

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: An out parameter allows an initiator to push an open-ended stream back to a responder, as specified in [C706] section 4.2.14.

Pipe: A supported **IDL** data type for streaming data, as specified in [C706] section 4.2.14.

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

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

[MS-RDPEDYC] Microsoft Corporation, "Remote Desktop Protocol: Dynamic Channel Virtual Channel Extension", June 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-SECO] Microsoft Corporation, "Windows Security Overview", January 2007.

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

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

1.2.2 Informative References

[MSDN-RPCMESSAGE] Microsoft Corporation, "RPC_MESSAGE", http://msdn2.microsoft.com/en-us/library/aa378631.aspx

1.3 Protocol Overview (Synopsis)

The Terminal Services Gateway Server Protocol is based on the Remote Procedure Call Over HTTP Protocol, as specified in [MS-RPCH]. 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.

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.

The TSG server maintains the TSG client state information. Communication from the TSG server to the TSG client occurs by using an RPC **out pipe** over the channel. Communication from the TSG client to the TSG server occurs by using RPC calls over the channel. All tunnel communication follows a simple request-response model: for every method call the TSG server receives, it executes and returns either the appropriate error code or data, as the method requires. All method calls are performed in a sequential order as shown in the diagram below and use asynchronous RPC calls.

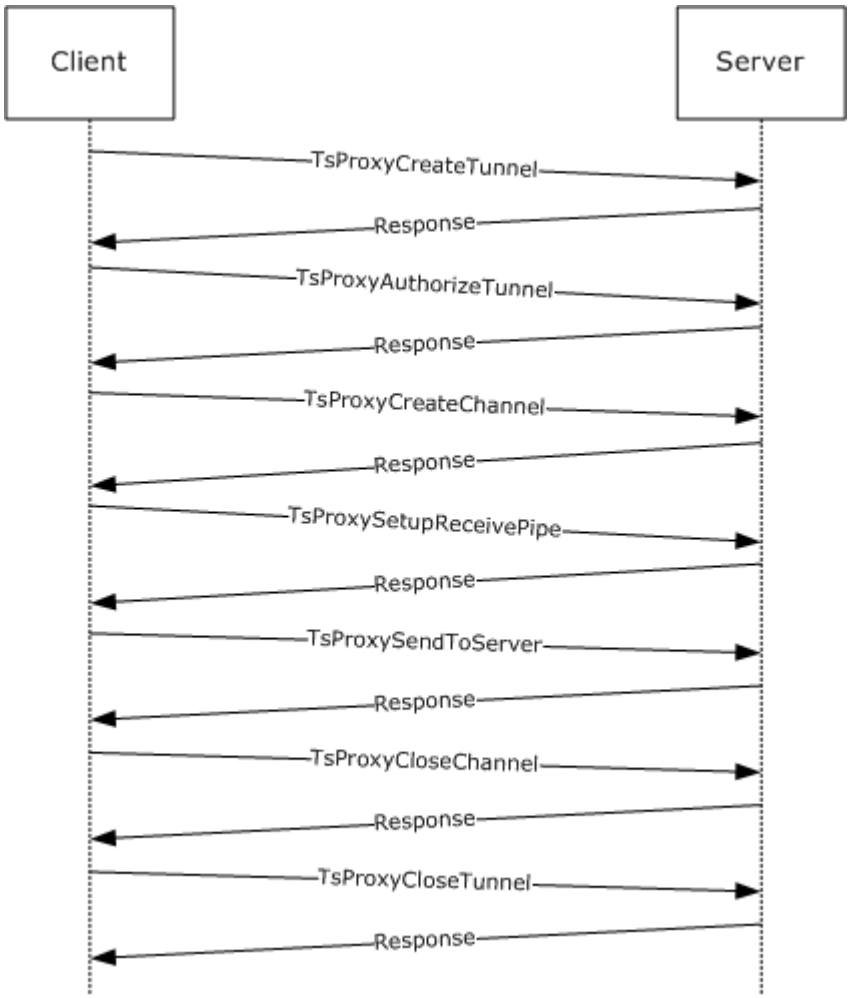


Figure 1: Method call sequence between the TSG client and TSG server

1.3.1 Role of the TS Gateway in a TS Connection

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 isolated target servers 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 TSG channel.

1.3.2 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.

1.3.2.1 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 3.1.4.3.2, 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.

- 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.

1.3.2.2 Data Transfer Phase

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

- TSG server to TSG client data transfer via an 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 from the TSG server to the TSG client is streamed via this **pipe**. The out pipe setup involves creating an asynchronous 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.

- TSG client to TSG server data transfer via RPC call: Data transfer from the TSG client to the TSG server is accomplished by using RPC method calls. The method call transfers data from the TSG client 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 single tunnel or channel within a single tunnel.

1.3.2.3 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 RPCout pipe created in the data transfer phase and prevents any further use of the channel. In normal operations, this is the target call for the channel in the protocol sequence. The closing of a channel is accomplished by using the **TsProxyCloseChannel** method call and follows the specification of a context **handle** closure, as specified in [MS-RPCE].

- Tunnel Shutdown: Tunnel shutdown can be performed only after a successful tunnel creation. 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 and follows the specification of a context handle closure as specified in [MS-RPCE].

1.4 Relationship to Other Protocols

This protocol is dependent upon the Remote Procedure Call Over HTTP Protocol (as specified in [MS-RPCH]) for its transport.

No other protocol currently depends on the Terminal Services Gateway Server Protocol. RDP uses the Terminal Services Gateway Server Protocol as its transport for traversing corporate firewalls. RDP is specified in [MS-RDPEDYC].

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 needs to connect to a 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 MUST return 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 [C706] section 2.7 and [MS-RPCE] section 1.7. In addition, it also supports the following authentication methods: NTLM, Schannel for the RPC server, Basic, NTLM, Schannel, and Integrated Windows Authentication for IIS. These **authentication services** are specified in [C706] section 13.1.2.2.

- 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 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
endpoint	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 RPC protocol sequences, as specified in [MS-RPCE] section 2.1.1 and [MS-RPCH] section 1.4.

This protocol uses the following static **endpoints** as well as **well-known endpoints**. These endpoints are ports for [MS-RPCH] section 1.5 and [MS-RPCE] section 2.1.1 on the TSG server. The only protocol sequence used for the transport is "ncacn_http".

- Port 443: This endpoint is used by [MS-RPCH] as the underlying transport.
- 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.

Both 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.

The TSG server MUST register for ipv4 and ipv6 localhost 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-SECO]. The TSG client MUST use a minimum authentication level of RPC_C_AUTHN_LEVEL_PKT_INTEGRITY (see [C706] section 13.1.2.1) 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 as specified in [MS-SECO].<1>

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 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 null-terminated name of a target server to which the TSG server connects. The name MUST not be null and SHOULD be a valid server name. The **RESOURCENAME** string MUST NOT exceed 512 Unicode characters. A **RESOURCENAME** can be either a list of IP addresses or FQDN or NetBIOS names.

2.2.1.2 PTUNNEL_CONTEXT_HANDLE_NOSERIALIZE

An RPC context handle representing the tunnel for the given connection.

This type is declared as follows:

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

This MUST be a non-serialized context handle.

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 3.1.5.3.2.2.2 and 3.3.1.1.2.

2.2.1.3 PCHANNEL_CONTEXT_HANDLE_NOSERIALIZE

An RPC context handle representing the channel for the given connection.

This type is declared as follows:

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

This MUST be a non-serialized context handle.

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 3.1.5.3.2.2.2 and 3.3.1.1.2.

2.2.1.4 PTUNNEL_CONTEXT_HANDLE_SERIALIZE

An RPC context handle representing the tunnel for the given connection.

This type is declared as follows:

```
typedef [context handle] PTUNNEL CONTEXT HANDLE NOSERIALIZE
PTUNNEL_CONTEXT_HANDLE_SERIALIZE;
```

This MUST be a serialized context handle.

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 3.1.5.3.2.2.2 and 3.3.1.1.2.

2.2.1.5 PCHANNEL_CONTEXT_HANDLE_SERIALIZE

An RPC context handle representing the channel for the given connection.

This type is declared as follows:

```
typedef [context handle] PCHANNEL CONTEXT HANDLE NOSERIALIZE
PCHANNEL CONTEXT HANDLE SERIALIZE;
```

This MUST be a serialized context handle.

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 3.1.5.3.2.2.2 and 3.3.1.1.2.

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 **tsgPacket** union field in the **TSG_PACKET** structure is a pointer to a **TSG_PACKET_VERSIONCAPS** structure.

2.2.2.4 TSG_PACKET_TYPE_QUARCONFIGREQUEST

Constant/value	Description
TSG_PACKET_TYPE_QUARCONFIGREQUEST	This constant is used by the **packetId** field of the **TSG_PACKET** structure. The TSG client SHOULD NOT use

Constant/value	Description
0x00005143	this constant on any method calls. The TSG server SHOULD reject any **TSG_PACKET** containing this structure type with an **HRESULT** value of E_PROXY_NOTSUPPORTED.

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 **tsgPacket** union field in the **TSG_PACKET** structure is a pointer to a **TSG_PACKET_QUARREQUEST** structure. It is also used by the 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 **tsgPacket** union field, in the **TSG_PACKET** structure, is 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 **tsgPacket** union field in the **TSG_PACKET** structure is 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 **NAP** capabilities are supported by the TSG client and TSG 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. These are the names of the target servers to which the TSG server can connect. The range is from 1 to **numResourceNames**. The TSG server SHOULD attempt to connect to each **RESOURCENAME** until any one of them succeeds. The TSG server MUST stop connecting to further servers specified by their given **RESOURCENAME**, after a successful connection has been made. The order in which a connection to a target server is attempted can be determined by the TSG server. The TSG client MUST specify at least one string for the **resourceName** array.

numResourceNames: The number of **RESOURCENAME** datatypes in the **resourceName** array. The TSG client MUST limit this to a maximum of 50 and a minimum of 1. The TSG server SHOULD limit this to a maximum of 50.

alternateResourceNames: An array of **RESOURCENAME** strings to be used as alternative names for the target server. The range is from 0 to **numAlternateResourceNames**. The TSG server MAY use these names for policy checks or connection attempts, or ignore them. The TSG client can specify 0 or more **alternateResourceNames** to be used in conjunction with **resourceName**.<2>

numAlternateResourceNames: The number of allowed **alternateResourceNames**. The TSG client MUST limit this to a maximum of three.

Port: Specifies the protocol ID and TCP port number for the target server endpoint to which the TSG server connects. The TSG client MUST specify the protocol ID in the low order 16 bits of this field and port number in the high order 16 bits. The TSG server MAY use these values to connect to the target server or ignore them. If the TSG server uses them, it MUST interpret the low order 16 bits as a protocol ID and interpret high order 16 bits as the actual port number. 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 contained in the **tsgPacket** field. Valid values are specified in sections 2.2.2.3, 2.2.2.5, 2.2.2.6, and 2.2.2.7.

tsgPacket: A union field containing the actual structure 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, and 2.2.3.2.1.6.

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;
} 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.

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: Specifies the id of the type of structure. Valid values are specified in sections 2.2.2.3, 2.2.2.5, 2.2.2.6, and 2.2.2.7. This is a redundant field and MAY be ignored by the TSG client and TSG server.

The TSG client and TSG server MUST NOT use this structure 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**.

The TSG client MUST send this structure to the TSG server as part of the **TsProxyCreateTunnel** method call. The TSG server MUST send this structure 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. The TSG server MAY ignore the values in this structure. The TSG server MUST send at least one **TSG_PACKET_CAPABILITIES** array element to the TSG client.

numCapabilities: The number of array elements for the **tsgCaps** field. The TSG client MUST limit this value to a maximum of 32, and SHOULD, at a minimum, specify one value. The TSG server SHOULD limit the value to 32 and MAY choose to ignore this value.

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:<3>

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

The TSG server will ignore TSG client's value. The TSG server MAY send a value of 1 to indicate quarantine policies enforcement to the TSG client<4>.

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. The TSG client uses this value to indicate NAP capabilities. The TSG server MUST send this packet to the TSG client.

```
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).<5>

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 2.2.3.2.1.2.1.1.1.

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.

2.2.3.2.1.2.1.1.1 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. The TSG client can use this structure to indicate NAP capabilities. The TSG server MAY choose to ignore this structure from the TSG client. The TSG server MUST send this structure to the TSG client with the following values.

```
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 can be the following value.

Value	Meaning
0x00000001	The TSG server supports the NAP capability of receiving and processing a quarantine statement of health from the TSG client.

2.2.3.2.1.3 TSG_PACKET_QUARCONFIGREQUEST

The TSG client and TSG server MUST NOT use this structure.

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

flags: This member is reserved for future use and MUST NOT be used by either the TSG client or the TSG server.

2.2.3.2.1.4 TSG_PACKET_QUARREQUEST

The **TSG_PACKET_QUARREQUEST** structure<6> contains information about the TSG client's statement of health 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.

The TSG client MUST send this structure to the TSG server. The TSG server MUST NOT send this structure to the TSG client.

```
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. The format of the machine name is determined by the TSG server policy. The TSG client SHOULD send its machine name to the TSG server. The TSG server MAY choose to ignore this value depending on server connection authorization policies set by the administrator <7>. The length of the name MUST be less than or equal to the size specified by the **nameLength** field.

nameLength: Length of the string specified by **machineName** including the null terminator. This MUST NOT exceed 513 Unicode characters.

data: An array, of bytes, specifying the statement of health. The TSG client SHOULD send this data to the TSG server. The TSG server MAY choose to ignore this data based on its policies.<8> The length of this data is specified by the **dataLen** field.

dataLen: Length, in bytes, of the **data** field. This value MUST NOT exceed 8000.

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. The TSG server MAY send a null data for this field, depending on its policy <9>. The TSG client MAY ignore this data. The length of the data MUST be equal to that specified by **responseDataLen**.<10>

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

redirectionFlags: A **TSG_REDIRECTION_FLAGS** structure. The TSG client MUST use the values sent by the TSG server.<11>

2.2.3.2.1.5.1 TSG_REDIRECTION_FLAGS

The **TSG_REDIRECTION_FLAGS** structure specifies the device redirection settings, as specified in [MS-RDPEDYC], that MUST be enforced by the TSG client.

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

```
typedef struct _TSG_REDIRECTION_FLAGS {
  long enableAllRedirections;
  long disableAllRedirections;
  long driveRedirectionDisabled;
  long printerRedirectionDisabled;
  long portRedirectionDisabled;
  long reserved;
  long clipboardRedirectionDisabled;
  long 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
0x00	The TSG client MUST process and honor the remaining fields in this structure and MUST NOT choose its own redirection settings.
0x01	The TSG client MUST ignore the remaining fields in this structure and MAY choose its own redirection settings.

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

Value	Meaning
0x00	The TSG client MUST process and honor the remaining fields in this structure and MAY choose its own redirection settings for some devices as indicated by the remaining fields.
0x01	The TSG client MUST ignore the remaining fields in this structure and disable all device redirection.

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

Value	Meaning
0x00	The TSG client MAY choose its own redirection settings for enabling or disabling drive redirection.

Value	Meaning
0x01	The TSG client MUST disable drive redirection.

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

Value	Meaning
0x00	The TSG client MAY choose its own redirection settings for enabling or disabling printer redirection.
0x01	The TSG client MUST disable printer redirection.

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

Value	Meaning
0x00	The TSG client MAY choose its own redirection settings for enabling or disabling port redirection.
0x01	The TSG client MUST disable port redirection.

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
0x00	The TSG client MAY choose its own redirection settings for enabling or disabling clipboard redirection.
0x01	The TSG client MUST disable clipboard redirection.

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

Value	Meaning
0x00	The TSG client MAY choose its own redirection settings for enabling or disabling PnP redirection.
0x01	The TSG client MUST disable PnP redirection.

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**.

The TSG client MUST NOT send this structure to the TSG server. The TSG server MUST send this structure to the TSG client.

```
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: Length, in bytes, of the **certChainData** field. The TSG server MUST provide a nonzero value for this field if the **quarantineCapabilities** field of the **TSG_PACKET_VERSIONCAPS** structure is set to 1. The TSG client MAY choose to ignore this value depending on its capabilities and policies.

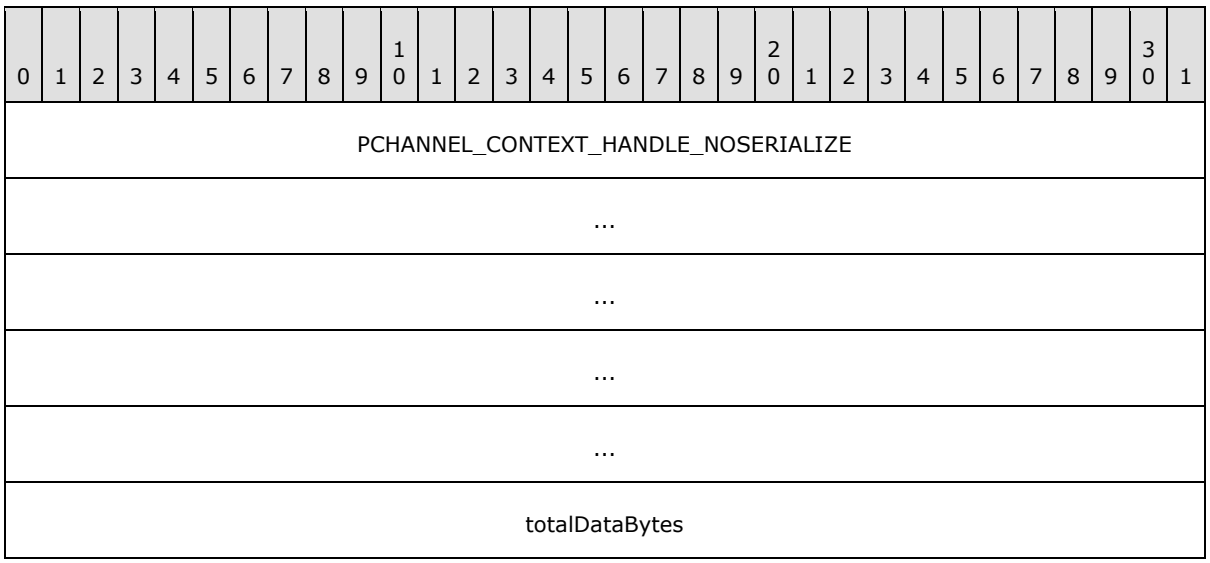
certChainData: The certificate, along with the chain, that the TSG server used for the SCHANNEL authentication service as part of registering the RPC interfaces and initialization. The TSG server MUST provide a string representation of the certificate chain if **certChainLen** is nonzero. The TSG client MAY choose to ignore this value depending on its capabilities and policies.<12>

nonce: A **GUID** to uniquely identify this connection to prevent replay attacks by the TSG client. TSG server MAY use this for auditing purposes. The TSG client MAY save this GUID for auditing purposes.

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

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. The TSG server SHOULD send this to the target server.



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

PCHANNEL_CONTEXT_HANDLE_NOSERIALIZE (20 bytes): This MUST be the wire representation of the **PCHANNEL_CONTEXT_HANDLE_NOSERIALIZE** data type returned by the TSG server by using the **TsProxyCreateChannel** method call.

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. The TSG client MUST limit this to a maximum of 3 and a minimum of 1.

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.

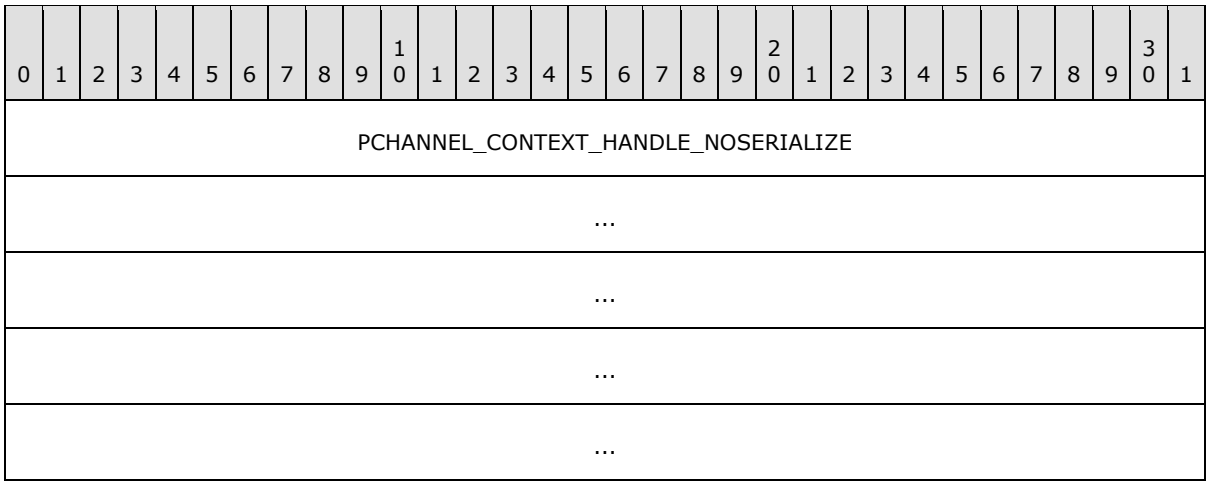
buffer2 (variable): The **buffer2** is an array of bytes. Its length is specified by **buffer2Length**. This MUST be non-NULL if buffer2Length is non-zero and contain the same number of bytes

specified by **buffer2Length**. If buffer2Length is 0, this SHOULD be NULL. 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 non-zero and contain the same number of bytes specified by **buffer3Length**. If buffer3Length is 0, this SHOULD be NULL. The contents of **buffer3** are opaque to the Terminal Services Gateway Server Protocol.

2.2.3.4 Generic Receive Pipe Message Packet

This packet contains data sent by the TSG client to the TSG server to set up the out pipe. This is sent by the TSG client for the **TsProxyCreateChannel** method call. The TSG server MUST use it to send all data from the target server to the TSG client.



PCHANNEL_CONTEXT_HANDLE_NOSERIALIZE (20 bytes): This MUST be the wire representation of the **PCHANNEL_CONTEXT_HANDLE_NOSERIALIZE** data type returned by the TSG server obtained by using the **TsProxyCreateChannel** method call. The TSG server MUST obtain the channel context using this wire representation of the first 20 bytes of this method and perform channel context-specific operations, including setting up the out pipe. The TSG client and TSG server MUST keep the out pipe valid as long as data exchange is required from the target server to the TSG client.

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 **HRESULTs** are specified by this protocol.<13>In the following paragraphs, 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 tunnel has been created. If tunnel has not been created, the client SHOULD close the binding handle.

Name (value)	Meaning
E_PROXY_INTERNALERROR (0x800759D8)	Used as a generic catch-all when an expected error happens, the TSG server sends this to the TSG client. The TSG client and TSG server MUST disconnect and end the protocol when this error is received.
E_PROXY_RAP_ACCESSDENIED (0x800759DA)	Returned when an attempt to resolve or access a target server is blocked by TSG server policies. The TSG client and TSG server MUST disconnect and end the protocol when this error is received.
E_PROXY_NAP_ACCESSDENIED (0x800759DB)	Returned when the TSG server denies the TSG client access due to policy. The TSG client and TSG server MUST disconnect and end the protocol when this error is received.
E_PROXY_TS_CONNECTFAILED (0x800759DD)	Returned when the target server cannot be reached. The TSG client and TSG server MUST disconnect and end the protocol when this error is received.
E_PROXY_ALREADYDISCONNECTED (0x800759DF)	Returned when an operation is called on a disconnected tunnel or channel. The TSG client and TSG server MUST end the protocol when this error is received.
E_PROXY_MAXCONNECTIONSREACHED (0x800759E6)	The TSG server has reached the maximum connections allowed. The TSG client and TSG server MUST disconnect and end the protocol when this error is received.
E_PROXY_NOTSUPPORTED (0x800759E8)	The TSG server does not support the requested version or capabilities. The TSG client and TSG server MUST disconnect and end the protocol when this error is received.
E_PROXY_QUARANTINE_ACCESSDENIED (0x800759ED)	The TSG server rejects connection due to quarantine policy. The TSG client and TSG server MUST disconnect and end the protocol when this error is received.
E_PROXY_NOCERTAVAILABLE (0x800759EE)	The TSG server cannot find a certificate to register for SCHANNEL authentication service. The TSG client and TSG server MUST disconnect and end the protocol when this error is received.

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.

User Name: A string of Unicode characters that can't exceed 512, including the NULL terminator. The TSG server SHOULD impersonate the TSG client user to determine the user name. The user name MAY be used to keep track of who successfully connected or failed to connect. This is useful for auditing purposes.

Target server name: A string of Unicode characters that can't exceed 512, including the NULL terminator.

Client Machine name: A string of Unicode characters that can't exceed 512, including the NULL terminator.

Tunnel id: An unsigned long representing the tunnel identifier for tracking purposes on the TSG server. It MAY be used by the TSG server as an index to access the tunnel context object.

Channel id: An unsigned long representing the channel identifier for tracking purposes on the TSG server. It MAY be used by the TSG server as an index to access the channel context object.

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 TSG client to TSG target server connection represented by an array of 20 bytes on the TSG server.

3.1.2 Timers

No protocol timer events are required on the TSG server beyond the timers required in the underlying RPC protocol.

3.1.3 Initialization

Parameters necessary to initialize the TSG server are described in section 2.1.

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 1.3.

The server SHOULD<14> 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 server MUST be sequential in order as specified in section 1.3.2. The method details are described below.

Methods in RPC Opnum Order

Method	Description
Opnum0NotUsedOnWire	Reserved for local use. Opnum: 0
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
Opnum3NotUsedOnWire	Reserved for local use. Opnum: 3
TsProxyCreateChannel	Creates a channel between the TSG client and the target 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 table above, the term "Reserved for local use" means that the client MUST NOT send the opnum, and the server behavior is undefined <15> since 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 on the condition there is no active channel. This is accomplished by using the **TsProxyCloseTunnel** method call.

```
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. The value of the **packetId** field MUST be set to **TSG_PACKET_TYPE_VERSIONCAPS**. The *tsgPacket* field MUST be a **TSG_PACKET_VERSIONCAPS** structure.

tsgPacketResponse: Pointer to the **TSG_PACKET** structure. The value of the **packetId** field MUST be set to **TSG_PACKET_TYPE_QUARENC_RESPONSE**. The *tsgPacket* field MUST be a **TSG_PACKET_QUARENC_RESPONSE** structure.

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 SHOULD provide this value to the TSG client. The TSG client SHOULD save the tunnel id for future use. This tunnel id is not required on any future method calls.

Return Values: The method MUST return ERROR_SUCCESS (0x00000000) on success. Other failures MUST be one of the codes listed. The client MAY interpret failures in any way it deems appropriate.

ERROR_SUCCESS (0x00000000)
E_PROXY_INTERNALERROR (0x800759D8)
E_PROXY_QUARANTINE_ACCESSDENIED (0x800759ED)
E_PROXY_NOCERTAVAILABLE (0x800759EE)

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.<16> After this method call has successfully been completed, a tunnel shutdown can be performed on the condition there is no active channel. If there are existing channels within the tunnel, all of them SHOULD be first closed using **TsProxyCloseChannel**. The tunnel shutdown is accomplished by using the **TsProxyCloseTunnel** method call.

```
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**. The *tsgPacket* field MUST be a **TSG_PACKET_QUARREQUEST** structure.

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

Return Values: The method MUST return ERROR_SUCCESS (0x00000000) on success. Other failures MUST be one of the codes listed. The client MAY interpret failures in any way it deems appropriate.

ERROR_SUCCESS (0x00000000)
E_PROXY_INTERNALERROR (0x800759D8)
E_PROXY_NAP_ACCESSDENIED (0x800759DB)
E_PROXY_ALREADYDISCONNECTED (0x800759DF)
E_PROXY_MAXCONNECTIONSREACHED (0x800759E6)
E_PROXY_NOTSUPPORTED (0x800759E8)
E_PROXY_QUARANTINE_ACCESSDENIED (0x800759ED)

3.1.4.1.3 TsProxyCreateChannel (Opnum 4)

The **TsProxyCreateChannel** method is used to create a channel between the TSG client and the target server.<17> The TSG server SHOULD connect to the target server during this call to start communication between the TSG client and target server<18>. 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.

```
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.

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 SHOULD provide this value to the TSG client. The TSG client SHOULD save the channel id for future use. This channel id is not required on any future method calls.

Return Values: The method MUST return ERROR_SUCCESS (0x00000000) on success. Other failures MUST be one of the codes listed. The client MAY interpret failures in any way it deems appropriate.

ERROR_SUCCESS (0x00000000)
E_PROXY_INTERNALERROR (0x800759D8)
E_PROXY_RAP_ACCESSDENIED (0x800759DA)

E_PROXY_ALREADYDISCONNECTED (0x800759DF)

3.1.4.2 Data Transfer Phase

3.1.4.2.1 TsProxySendToServer (Opnum 9)

The **TsProxySendToServer** method is used for data transfer from the TSG client to 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] 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. The TSG server MUST handle multiple invocations of this method call. It is the responsibility of the RPC runtime to correctly return the number of bytes in the array to the caller.<19>

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

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

Return Values: The method MUST return ERROR_SUCCESS (0x00000000) on success. Other failures MUST be one of the codes listed. The client MAY interpret failures in any way it deems appropriate.

ERROR_SUCCESS (0x00000000)
E_PROXY_INTERNALERROR (0x800759D8)
E_PROXY_TS_CONNECTFAILED (0x800759DD)
E_PROXY_ALREADYDISCONNECTED (0x800759DF)

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 asynchronous 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. The out pipe must be created by the TSG server in the same manner as NDR creates it for an asynchronous call.<20> The TSG server MUST use this out pipe 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. The last chunk of data from the TSG server is indicated when a 0-byte datum is sent from the TSG server to the TSG client. This is followed by a DWORD return code. 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. The TSG server SHOULD create the connection to the target server by the time this method is completed. It is the responsibility of the RPC runtime to correctly return the number of bytes in the array to the caller.<21>

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

```
);
```

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

Return Values: The method MUST return ERROR_SUCCESS (0x00000000) on success. Other failures MUST be one of the codes listed. The client MAY interpret failures in any way it deems appropriate. The E_PROXY_TS_CONNECTFAILED error code is typically returned when this call fails before the first **TsProxySendToServer** is completed by the server. This happens in race conditions and is not easily seen.

ERROR_SUCCESS (0x00000000)
E_PROXY_INTERNALERROR (0x800759D8)
E_PROXY_TS_CONNECTFAILED (0x800759DD)
E_PROXY_ALREADYDISCONNECTED (0x800759DF)

3.1.4.3 Shutdown Phase

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. 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.

```
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: The method MUST return 0 on success. This function SHOULD NOT fail from a TSG protocol perspective.

ERROR_SUCCESS (0x00000000)

3.1.4.3.2 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 there are existing channels within the tunnel, all of them SHOULD be first closed using **TsProxyCloseChannel**.

```
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.

ERROR_SUCCESS (0x00000000)

3.1.5 Timer Events

No protocol timer events are required on the server beyond the timers required in the underlying RPC transport.

3.1.6 Other Local Events

There are no local events used on the TSG server beyond the events maintained in the underlying RPC transport.

3.2 TsProxyRpcInterface Client Details

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 that can't exceed 512, including the NULL terminator.

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.

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. It MAY contain data about the user name, the TSG client machine name, and the TSG client health information.

Channel Context handle: An RPC context handle for the TSG client to TSG target server connection represented by an array of 20 bytes on the TSG server. It MAY contain data about the target server name and port number that the TSG client has connected.

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.

3.2.2 Timers

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 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 asynchronous calls.

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 1.3.2. The method details are described 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. Otherwise, no 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

No protocol timer events are required on the client beyond the timers required in the underlying RPC transport.

3.2.6 Other Local Events

There are no local events used on the client beyond the events maintained in the underlying RPC transport.

4 Protocol Examples

1. The TSG client obtains the name of a TSG server by using an out-of-band mechanism. The TSG client establishes a binding handle to the TSG server at the well-known endpoint of 443 and 3388. The TSG server performs the authentication steps described in section 2.1. The TSG client then calls the **TsProxyCreateTunnel** method to create and obtain the tunnel context handle.

2. 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.

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

4. 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.

5. 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.

6. 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.

7. The TSG client makes the **TsProxySetupReceivePipe** method call. The TSG client uses the channel context handle to set up the out pipe to be able to receive data from the target server.

8. The TSG server receives the **TsProxySetupReceivePipe** method and creates a RPC asynchronous pipe. The TSG server can now send data on the pipe.

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

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 = 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_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 = 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;
```

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 = NULL;
   unsigned long dataLen = 0;
} 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 **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* sessionId = { 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* sessionId  = 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:

```
WireRepresentation of 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
{
   WireRepresentation of m_channelContextHandle; //as above
   totalDataLength = 08000000; //buffer1Length+sizeof(buffer1Length)
   numBuffers = 01000000; //number of buffers that follow is 1
   buffer1Length=04000000; //length of data that follows is 4 bytes
   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] WireRepresentation  of m_channelContextHandle as above);
 );
```

The TSG server receives this method, sets up the async out pipe, streams all necessary data to the client, and after the last chunk of data has been sent (indicated by a 0-byte data push through the out pipe), returns the following.

```
DWORD = ERROR_SUCCESS
TsProxySetupReceivePipe (
   [in] WireRepresentation  of m_channelContextHandle = 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;
);
```

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 replay attacks by the TSG client. The TSG client MUST save this GUID for future use.

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

    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 {
    long enableAllRedirections;
    long disableAllRedirections;
    long driveRedirectionDisabled;
    long printerRedirectionDisabled;
    long portRedirectionDisabled;
    long reserved;
    long clipboardRedirectionDisabled;
    long 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 [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;
} 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
);
```

```
     void Opnum3NotUsedOnWire(void);

    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: Windows Behavior

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

- Windows XP SP2
- Windows Server 2003 SP1
- Windows Vista
- Windows Server 2008

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

<1> Section 2.1: 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_IMP_LEVEL_IDENTIFY on all method calls. If TSG 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 using the same certificate that is set for HTTPS communications on the machine.

<2> Section 2.2.3.1: Windows Server 2003 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.

<3> Section 2.2.3.2.1.2: Windows XP SP2, Windows Vista, Windows Server 2003 SP1, and Windows Server 2008 send capability 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.

<4> Section 2.2.3.2.1.2: Windows Server 2008 sends capability type 1—indicating that the server 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.

<5> Section 2.2.3.2.1.2.1: Windows XP SP2, Windows Vista, Windows Server 2003 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.

<6> 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.

<7> 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.

<8> Section 2.2.3.2.1.4: Windows XP SP2, Windows Vista, and Windows Server 2003 SP1 obtain the health statement from the NAP agent and encrypt it using the certificate sent by the server during the **TsProxyCreateTunnel** method. Windows Server 2008 decrypts the health statement 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.

<9> Section 2.2.3.2.1.5: Windows Server 2008 sends a NULL.

<10> Section 2.2.3.2.1.5: Windows Server 2008 sends a NULL.

<11> Section 2.2.3.2.1.5: Windows Server 2008 sends the **redirectionFlags** value based on network policies at the Windows terminal server.

<12> 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.

<13> Section 3.1: All TSG-supported versions of Windows use the identity of the caller to perform method-specific access checks. The TS Gateway service allows only authenticated users to call any method. The Windows Server 2008 imposes a minimum impersonation level of RPC_C_IMP_LEVEL_IDENTIFY on all method calls. If the TS Gateway is operating in a load-balanced environment, the Windows Server 2008 registers for the hostname, not 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.

<14> 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.

<15> Section 3.1.4: Opnums reserved for local use apply to Windows XP SP2, Windows Vista, Windows Server 2003 SP1, and Windows Server 2008 as follows.

Opnum	Description
0	The server has deprecated code for an unused method.
3	Not used by all versions of Windows.
5	Not used by all versions of Windows.

<16> Section 3.1.4.1.2: When the connection authorization policies set by the administrator require health checks for TSG client, the TSG server will require that TSG clients send health information and remediate themselves if health check is not met.

<17> Section 3.1.4.1.3: 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.

<18> Section 3.1.4.1.3: Windows Server 2008 does not attempt to connect to the target end point during the TsProxyCreateChannel call. The actual connection to the end point happens during the call to TsProxySetupReceivePipe.

<19> Section 3.1.4.2.1: Windows Server 2008, Windows Server 2003 SP1, Windows XP SP2, and Windows Vista do not use the NDR 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 waits for up to 30 seconds if a connection to the target endpoint has not yet been completed. After this time, a connection failure is returned to Windows Server 2003 SP1, Windows XP SP2, or Windows Vista client.

<20> Section 3.1.4.2.2: To bypass NDR, the 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].

<21> Section 3.1.4.2.2: Windows Server 2008, Windows Server 2003 SP1, Windows XP SP2, and Windows Vista do not use the NDR for this call. Windows Server 2003 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.

8 Index